Oracle® Database Lite

Troubleshooting and Tuning Guide

10*g* (10.3.0)

B28927-01

April 2007

ORACLE®

Oracle Database Lite Troubleshooting and Tuning Guide 10*g* (10.3.0)

B28927-01

Contents

2 Troubleshooting

3 Tracing and Logging

4 Backup and Recovery

Index

Preface

This preface introduces you to the *Oracle Database Lite Developer's Guide*, discussing the intended audience, documentation accessibility, and structure of this document.

Audience

This manual is intended for application developers as the primary audience and for database administrators who are interested in application development as the secondary audience.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Send Us Your Comments

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: olitedoc_us@oracle.com
- FAX: (650) 506-7355. Attn: Oracle Database Lite
- Postal service:

  Oracle Corporation
  Oracle Database Lite Documentation
  500 Oracle Parkway, Mailstop 1op2
  Redwood Shores, CA 94065
  U.S.A.

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

1 Improving Performance

The following sections describe the methods you can manage the performance of your use of Oracle Lite Database:

- Section 1.1, "Improving Connection Performance"
- Section 1.2, "Increasing Synchronization Performance"
- Section 1.3, "Determining Performance of Client SQL Queries With the EXPLAIN PLAN"
- Section 1.4, "Optimizing Application SQL Queries Against Oracle Lite Database"
- Section 1.5, "Maximizing JVM Performance By Managing Java Memory"

1.1 Improving Connection Performance

The following methods enable you to streamline the connections between the client/server and the Mobile Server and back-end database:

- Section 1.1.1, "Using Connection Pooling for Applications"
- Section 1.1.2, "Limit Application Connection Requests to the Database"

1.1.1 Using Connection Pooling for Applications

Connection pooling enables you to eliminate the time delay in creating and destroying connections for incoming application requests. Instead, enable connection pooling, as shown in Section 3.4, "Manage Application Properties or Users" in the *Oracle Database Lite Administration and Deployment Guide*, so that each incoming connection request uses an existing connection from the pool.

1.1.2 Limit Application Connection Requests to the Database

You can limit the number of connections that access the database from each application, as shown in Section 3.4, "Manage Application Properties or Users" in the *Oracle Database Lite Administration and Deployment Guide*. Set the maximum database connection limit. Any request for a database connection beyond the limit is refused.

1.2 Increasing Synchronization Performance

The following sections describe how you can manipulate the synchronization performance:

- Section 1.2.1, "Analyzing Performance of Publications With the Consperf Utility"
- Section 1.2.2, "Monitoring Synchronization Using SQL Scripts"

1.2.1 Analyzing Performance of Publications With the Consperf Utility

The Consperf utility profiles your subscriptions and may modify how the publication item is executed if the utility determines that there is a more performant option. The Consperf tool evaluates how the SQL within the publication item interacts with our Data Synchronization query templates. The first synchronization is always a complete refresh, which is a direct invocation of the query. On subsequent synchronizations, the query templates determine incremental refreshes. This improves your performance from not having to perform a complete refresh each time you synchronize. However, the interaction of our query templates and your SQL may not be optimal, which is discovered by the Consperf tool. We either modify the query template or type of logical delete or insert for you or you can adjust your SQL to be more performant in regards to our templates.

In addition, application developers and administrators use this utility to analyze the performance of subscriptions and identify potential bottlenecks during synchronization.

This tool generates the following two primary analysis reports:

1. Timing statistics for publication items
2. Explain plans for publications

The Consperf tool automatically tunes subscription properties, if the default templates do not supply the highest performing option. You can select a client and choose the desired subscription for performance analysis. Users can change parameter values before analyzing performance. The analysis results, which are timing and execution plan reports, are stored on the server and can be accessed by viewing the same user and subscription.

You can execute the Consperf utility through one of the following locations:

- Click the Users link under the Consperf section on the Performance tab.

- Click the Users link from the Repository screen.

Then, perform the following:

1. Select the User that you want to execute the Consperf tool against and click **Subscriptions**.
2. From the subscriptions screen, choose the publication and click **Consperf performance analysis**. This starts the Consperf analysis.
3. Click **Set consperf parameters and launch the consperf thread**, which brings you to a screen where you can configure parameters that effect how the performance analysis is executed. See Section 1.2.1.1, "Deciphering the Performance Evaluation Files" for more information on these parameters and how they effect the performance evaluation output.
4. Once you have set the configuration for how you want your performance analysis to occur, click **OK**. The Consperf tool executes and prepares the reports for you, based on your configuration. You are returned to the first Consperf page with the reports listed as hyperlinks under the **Last Consperf Run Results** section as **View Timing File** or **View Execution Plan File**.

1.2.1.1 Deciphering the Performance Evaluation Files

There are two performance evaluatons that come out of the Consperf utility:

- Timing File
- Execution Plan File

Timing File

The timing file contains the analysis of how the publication item performs with the data synchronization defaults against how it could perform if other options were chosen. The output of this file shows you the conclusions of the analysis and how the data synchronization defaults could be modified to perform better with your particular publication items.

The first section of the timing file provides you information on the configuration with which this analysis was executed. Thus, if you modify the configuration for other analysis, you can go back and compare each file to each other to easily see the differences in the output.

> **Note:** The results of this analysis may cause the data synchronization engine to modify the type of query template or logical delete/insert/update used with your publication item. To change it back to the defaults, you will have to rerun Consperf with `CLEARTUNE` set to `YES`. See Table 1–2 for a full description of parameter settings.

The following example shows the publication that is examined is the `T_SAMPLE11` publication. The version of the Oracle Database Lite is 10.0.0.0.0. The user is `S11U1`. And the configuration is set to time out if the query takes longer that 1000 milliseconds and change the defaults if the difference between the default and the other templates are greater than 20 seconds (20000 milliseconds). The command that authorizes the changes is when `AUTOTUNE` is set to true. If set to false, the analysis is provided, but nothing is modified.

```
VERSION = 10.0.0.0.0
OPTMIZER_MODE = null
```

```
APPLICATION = null
PUBLICATION = T_SAMPLE11
CLIENTID = S11U1
TIMEOUT = 1000 ms
TOLERANCE = 20000 ms
ITERATIONS = 2
AUTOTUNE_SUPPORT = true
```

The next part of the Timing File lists the time in milliseconds each template type takes to complete with each publication item in the publication. There are three templates that data synchronization can use to "wrap" your SQL query. The default query template is SYNC_1. Since the tolerance is set to 20 seconds, then if either template SYNC_2 or SYNC_3 perform at least 20 seconds bettern than SYNC_1, then the template type will be modified for your publication item. You can set the TOLERANCE level to fewer seconds in the Consperf configuration. See Table 1–2 for a description of TOLERANCE.

```
Publication Item Name | NS | BS | SYNC_1 | SYNC_2 | SYNC_3 | AS | Total
 ---------------------------------------------------------------------
P_SAMPLE11-D          | <3> | <0> | <6>  | 10     | -1000  | <0> | 9
P_SAMPLE11-M          | <3> | <0> | <5>  | 8      | -1000  | <0> | 8
```

- There are two publication items in the subscription.
- NS stands for Null Sync. Your application may be issuing a null synchronization. If so, this shows the time in milliseconds that it took to complete. The null synchronization is a tool to see if it is the data that is causing the performance hit or the application itself.
- BS stands for Before Synchronization; AS stands for After Synchronization. You can provide callouts that are executed either before or after each synchronization for this application. This shows the time in milliseconds it takes to perform each task. In this example, there is no before or after synchronization callouts.
- SYNC_1 is the default template. In combination with the publication items, it still is executing the fastest as compared to the other two options: SYNC_2 and SYNC_3 with 6 and 5 milliseconds for each publication item respectively. Thus, these publication items will continue to use SYNC_1 template. Note that SYNC_3 has -1000 as its time. That either means that the template was not appropriate to execute or that it timed out.
  - SYNC_1 uses an outer-join for inserts, updates, and deletes
  - SYNC_2 is a simple insert and update
  - SYNC_3 uses the base view for insert and update. The base view is the first table in the select statement, as it is the primary key used to search for all records in the query.
- The total is the total number of milliseconds to execute the entire publication item.

The second section is how the MGP performs with the templates it uses for deletes and inserts. It evaluates the default against other options, as follows:

- Logical delete options:
  - MGP template for logical deletes using EXISTS: default for logical delete
  - MGP template for logical deletes using correlated IN
  - MGP template for logical deletes using HASH_AJ
  - MGP template for logical deletes using IN

- Logical insert options:
  - MGP template for logical inserts using EXISTS: default for logical insert
  - MGP template for logical inserts using correlated IN
  - MGP template for logical inserts using IN
- Logical update options
  - MGP template for logical updates using correlated IN: default for logical updates
  - MGP template for logical updates using EXISTS
  - MGP template for logical updates using IN
- MGP template for logical updates with multiple table dependencies

For example, the following evaluates how each publication item performs with its logical deletes:

```
MGP Output...
Pub Item Name | LDEL_1 | LDEL_2 | LDEL_3 | LDEL_4
P_SAMPLE11-D  | <5>    | 3      | 3      | 3
P_SAMPLE11-M  | <5>    | 3      | 5      | 4
```

The `LDEL_1` is the default and even though `LDEL_2` , `3` and `4` are faster, they are not 20 seconds faster, which is the tolerance level. So, the default for deletes is kept the same. If the difference in speed had been greater than the tolerance level, the Consperf utility would have modified the logical delete method in the repository for the publication item in future—if the autotune parameter was set to yes.

The last section, Subscription Properties, describes the following:

- Profiled: Has autotune been turned on and Consperf executed previously on this subscription?
- Base View: True if this publication item uses more than one table.
- How many records are in the subscription.
- How many records are dirty?
- How many records have been flagged as dirty to simulate an actual run? Up to the number of records in the subscription or MAXLOG will be flagged as dirty, whichever is least.

Configuration for Data Synchronization

Table 1–1 Consperf Parameters for Both Synchronization and MGP Processing

Parameter	Default Value	Allowed Values	Description
PUBITEMLIST	<ALL>	Pub1, Pub2, and so on.	Specifies comma-separated list of publication items to process. The default is all publication items in the publication.
SKIPPUBITEMLIST	<NONE>	Pub1, Pub2, and so on.	Specifies comma-separated list of publication items to skip.
OPTIMIZER	<DB>	Can set to RULE or CHOOSE; otherwise sets to what database is set to.	Specifies the optimizer mode to use within Oracle. The default is the current DB setting.

Table 1–1 (Cont.) Consperf Parameters for Both Synchronization and MGP Processing

Parameter	Default Value	Allowed Values	Description
ORDERBYPUBITEM	NO	Yes or No	Orders all output by publication item name.

Table 1–2 Consperf Parameters for Synchronization Timing Performance

Parameter	Default Value	Allowed Values	Description
TIMEOUT	10 seconds	Integer for seconds	Specifies the query timeout value in seconds. This is the amount of time Consperf will wait before it cancels a query.
UPDATECOUNT	5	Integer for number of records	Specifies the number of records to mark as dirty during synchronization.
MAXLOG	5000	Integer for number of records	Specifies the number of records to put in the log table. Simulates the transaction log
AUTOTUNE	NO	Yes or No	Enables auto-tune.
CLEARTUNE	NO	Yes or No	Clears existing auto-tune results.
TOLERANCE	20 seconds	Integer for seconds	A template must be faster by this number of seconds before it replaces the default template.

Execution Plan File

The exeuction plan file shows how your publication items interact with the different logical delete, insert, and update templates. From this report, you can evaluate your SQL to see if you want to modify it in any way to speed up your query. Set the optimizer parameter to designate how the database is organized. If you set this parameter to a setting that the database is not set to, it still acts as if the database is set to this way to show you how it would execute. See Table 1–3 for all configuration parameters that relate to this search.

Table 1–3 Consperf Parameters for Execution Performance Plan

Parameter	Default Value	Allowed Values	Description
GATHERSTATS	NO	Yes or No	Gathers optimizer statistics on all mobile server objects. MGP compose MUST be disabled while Consperf analyzes objects. Consperf blocks this automatically, but the safest approach is to manually stop the MGP process before running Consperf with the GATHERSTATS option. If Consperf fails while gathering statistics, users must re-run CLEARSTATS before starting the MGP process again.
CLEARSTATS	NO	Yes or No	Removes optimizer statistics on mobile server objects.
SQLTRACE	NO	Yes or No	Enables Oracle sql trace. TKPROF can be used to analyze the resulting trace file.

1.2.2 Monitoring Synchronization Using SQL Scripts

If, instead of viewing MGP statistics within the Mobile Manager, you would rather execute SQL scripts to monitor Mobile application status during synchronization, you may use any of the following SQL scripts to retrieve the desired information.

1.2.2.1 Synchronization Times for All Clients

Using the following script, you can check the latest successful synchronization times for all clients by retrieving such information from the `all_clients` table.

```
select client, lastrefresh_starttime, lastrefresh_endtime
from cv$all_clients
order by client
/
```

1.2.2.2 Failed Transactions for all Clients

Using the following script, you can retrieve a list of failed transactions for all clients from the `all_errors` table.

```
select client, transaction_id, item_name, message_text
from cv$all_errors
where message_text is not null
order by client,transaction_id
/
```

1.2.2.3 Completely Refreshed Publication Items for all Clients

Using the following SQL script, you can retrieve a list of all publication items for all clients which were completely refreshed during the last synchronization process.

```
select clientid, publication_item
from c$complete_refresh_log
order by clientid, publication_item
/
```

1.2.2.4 Publications Flagged for Complete Refresh for All Clients

Using the following SQL script, you can retrieve a list of publications for all clients that are flagged for a complete refresh during the next synchronization process.

```
select clientid, template as publication
from c$all_subscriptions
where crr = 'Y'
/
```

1.2.2.5 Clients and Publication where Subscription Parameters are Not Set

Using the following SQL script, you can retrieve a list of clients and their publications where the subscription parameters have not been set.

```
select client, name as publication, param_name, param_value
from cv$all_subscription_params
where param_value is null
order by client, name
/
```

1.2.2.6 Record Counts for Map-based Publication Item by Client

Using the following script, you can retrieve record counts for all clients in queues for map-based publication items, that are grouped by clients.

```
select clid$$cs as client, count(*) as "RECORD COUNT"
from c$in_messages
group by clid$$cs
/
```

1.2.2.7 Record Count for Map-based Publication Items by Store

Using the following SQL script, you can retrieve record counts for all client in-queues for map-based publication items, that are grouped by store.

```
select clid$$cs as client, tranid$$ as transaction_id, store as item_name,
count(*) as "RECORD COUNT"
from c$in_messages
group by clid$$cs, tranid$$, store
/
```

1.2.2.8 All Client Sequence Partitions and Sequence Values

Using the following SQL script, you can retrieve a list of all client sequence partitions and current sequence values.

```
select clientid, name, curr_val, incr
from c$all_sequence_partitions
order by clientid, name
/
```

1.2.2.9 All Publication Item Indexes

Using the following SQL script, you can retrieve a list of all publication item indexes.

```
select publication as NAME, publication_item, conflict_rule as "INDEX_TYPE",
columns
from c$all_indexes
order by publication, publication_item
/
```

1.2.3 Create SQL Scripts With All Dependencies

When you create a SQL script in MDW or with the Consolidator APIs, you should include all dependent DDL statements in the same script in the order necessary. If you separate dependent DDL statements into separate scripts, Oracle Database Lite may be executing them randomly, causing dependency errors and re-execution of each script. See Section 5.7 "Create and Load a SQL Script" in the *Oracle Database Lite Developer's Guide* for more information.

1.2.4 Configuration Parameters in the WEBTOGO.ORA that Affect Synchronization Performance

The following parameters in the `[CONSOLIDATOR]` section of the `webtogo.ora` file are used for tuning synchronization:

- `MAX_THREADS`

  The `MAX_THREADS` parameter is used by the MGP and controls the number of concurrent threads. As a rule, do not set this higher than 1.5 times the number of CPUs on the database machine. For example, if your system has four CPUs, theb you should not set it higher than six.

- `MAX_CONCURRENT`

  The `MAX_CONCURRENT` parameter controls how many users can synchronize in parallel. Once the maximum number of concurrent synchronization requests is reached, additional requests block until one or more synchronization requests completes. If you do not set this parameter, then there is no maximum.

- `CONNECTION_TIMEOUT`

  The `CONNECTION_TIMEOUT` parameter specifies in minutes the JDBC connection timeout for the synchronization session.

- `COMPOSE_TIMEOUT`

  The `COMPOSE_TIMEOUT` parameter specifies in seconds the MGP timeout for the compose phase for each user.

- `CONNECTION_POOL`

  The `CONNECTION_POOL` parameter enables pooling of database connections.

- `MAX_THREADS`

  The `MAX_THREADS` parameter sets the maximum number of threads spawned within the MGP process.

For full details on these and more parameters, see Section A.6, "CONSOLIDATOR" in the *Oracle Database Lite Administration and Deployment Guide*.

Each synchronization request requires a number of system resources, such as creating a database connection, using memory, and so on. If you have too many requests competing for the same resources, then the overall performance can be poor. Limiting the number of parallel requests with the `MAX_THREADS` and `MAX_CONCURRENCY` parameters improve the average response time.

Set the `MAX_THREADS` and `MAX_CONCURRENCY` parameters if you notice that the synchronization performance is not linear. For example, if twice the number of parallel requests results in a synchronization time that is five times longer for each client, then you probably have resource contention. The value depends on your environment and should be determined on a trial and error basis.

1.2.5 Tuning Queries to Manage Synchronization Performance

You can increase synchronization performance by monitoring the performance of the SQL queries in your applications. The following sections provide details on how to tune your queries:

- Section 1.2.5.1, "Avoid Using Non-Mergable Views"
- Section 1.2.5.2, "Tune Queries With Consperf Utility"

- Section 1.2.5.3, "Manage the Query Optimizer"

1.2.5.1 Avoid Using Non-Mergable Views

You should avoid using database query constructs that prevent a view from being mergable, as publication item queries that use non-mergable views do not perform well. Examples of such constructs are union, minus, and connect by. For more information on mergable views, see the Oracle database documentation.

1.2.5.2 Tune Queries With Consperf Utility

Once you have defined your application, use the `consperf` utility to profile the performance of the publication item queries. Mobile Server does not execute your publication item queries directly; instead the query is wrapped into a template query, which is executed by Mobile Server. The template query may have an unexpected query execution plan, resulting in poor performance. The `consperf` utility generates an EXPLAIN PLAN execution plan for those template queries, allowing you to tune your publication item query for best performance. In addition, consperf generates timing information for the execution of all template queries, so that you can identify bottleneck queries. For more information on the consperf utility, see Section 1.2.1, "Analyzing Performance of Publications With the Consperf Utility".

1.2.5.3 Manage the Query Optimizer

You must make sure that the optimizer picks the correct execution path when you either are using the cost-based optimizer or you have set the optimizer settings to `choose`. The optimizer can pick the correct execution path only when all of the tables are properly analyzed and statistics have been gathered for these tables.

The Mobile Server uses temporary tables during synchronization. Once a number of users have been created, and they have synchronized with Mobile Server, run consperf with the `gatherstats` option to generate the statistics information for the temporary tables. For more information on the consperf utility, see Section 1.2.1, "Analyzing Performance of Publications With the Consperf Utility".

1.2.6 Synchronization Tablespace Layout

Tablespace layout across multiple disks can improve the performance of Mobile Server data synchronization, as it reduces movement of the disk heads and improves I/O response time.

By default, the synchronization tablespace is `SYNCSERVER`, and is stored in the `mobile`*`xx`*`.dbf` file in the default location for the database instance under *`ORACLE_HOME`*, where *`xx`* is a number between 1 and 25. The tablespace name, filename, and file location for the tablespace is defined in the `$ORACLE_HOME/Mobile/Server/admin/consolidator_o8a.sql` script file, which is executed during the Mobile Server installation process. So, if you want to modify the tablespace, perform the following BEFORE you install the Mobile Server; otherwise, the default tablespace is created.

If you want to customize the `SYNCSERVER` tablespace, for example, by using multiple data files spread across several disks, or by using specific storage parameters, then you can precreate the `SYNCSERVER` tablespace with the required settings. The installation process automatically detects that the tablespace exists and uses it. Refer to the Oracle Database documentation for full details on how to create a tablespace.

1.2.7 Shared Maps

It is very common for publications to contain publication items that are used specifically for lookup purposes. That is, a publication item that creates a read-only snapshot. The server may change these snapshots, but the client would never update them directly. Furthermore, many users often share the data in this type of snapshot. For example, there could be a publication item called `zip_codes`, which is subscribed to by all Mobile users.

The main function of Shared Maps is to improve scalability for this type of publication item by allowing users to share record state information and reduce the size of the resulting replication map tables. By default, if you have a non-updatable publication item, it defaults to using shared maps.

> **Note:** Shared Maps can also be used with updatable snapshots if the developer is willing to implement their own conflict detection and resolution logic; however, normally shared maps are only for non-updatable snapshots.

Shared maps shrink the size of map tables for large lookup publication items and reduce the MGP compose time. Lookup publication items contain read-only data that is not updatable on the clients and that is shared by multiple subscribed clients. When multiple users share the same data, their query subsetting parameters are usually identical.

For example, a query could be the following:

```
SELECT * FROM  WHERE EMP WHERE DEPTNO = :dept_id
```

In the preceding example, all users that share data from the same department have the same value for `dept_id`. The default sharing method is based on subscription parameter values.

In the following example, the query is:

```
SELECT * FROM WHERE EMP WHERE DEPTNO = ( SELECT DEPTNO FROM
                 EMP WHERE EMPNO = :emp_id )
```

In this example, users from the same departments still share data. Their subsetting parameters are not equal, because each user has a unique `emp_id`. To support the sharing of data for these types of queries (as illustrated by the example), a grouping function can be specified. The grouping function returns a unique group `id` based on the client `id`.

There is also another possible use for shared maps. It is possible to use shared maps for shared updatable publication items. However, this type of usage requires implementation of a custom **dml** procedure that handles conflict resolution.

1.2.7.1 Performance Attributes

The performance of the MGP compose cycle is directly proportional to the following:

```
NC * NPI
```

where:

- `NC` = number of clients
- `NPI` = number of publication items that must be composed

With shared maps, the length of the MGP cycle is proportional to the following:

```
NC*(NPI - NSPI)  + NG*NSPI
```

where:

- `NSPI` = number of shared publication items
- `NG` = number of groups

> **Note:** If `NG = NC`, then the MGP performance is similar in both cases. However, with fewer groups and more shared publication items, the MGP compose cycle becomes faster. In addition, map storage requirements are governed by these same factors.

1.2.7.2 Shared Map Usage

To set up a publication item to be shared, use the `AddPublicationItem` API and enable the shared flag. It is also possible to toggle the shared property of a publication item once it is added to the publication with the `SetPublicationItemMetadata` API. Both the `AddPublicationItem` API and the `SetPublicationItemMetadata` API allow users to specify a PL/SQL grouping function. The function signature must be as follows:

```
(
CLIENT in VARCHAR2,
PUBLICATION in VARCHAR2,
ITEM in VARCHAR2
)return VARCHAR2.
```

The returned value must uniquely identify the client's group. For example, if client **A** belongs to the group **GroupA** and client **B** belongs to the group **GroupB**, the group function **F** could return:

```
F ('A','SUBSCRIPTION','PI_NAME') = 'GroupA'
F ('B','SUBSCRIPTION','PI_NAME') = 'GroupB'
```

The implicit assumption of the grouping function is that all the members of the **GroupA** group share the same data, and that all the members of the **GroupB** group share the same data.. The group function uniquely identifies a group of users with the same data for a particular **PUBLICATION ITEM**.

For the query example in Section 1.2.7, "Shared Maps", the grouping function could be:

```
Function get_emp_group_id (
  clientid in varchar2,
  publication in varchar2,
  item in varchar2
) return varchar2 is
  group_val_id varchar2(30);
begin
  select DEPTNO into group_val_id
  from EMP where EMPNO = clientid ;
  return group_val_id;
end;
```

> **Note:** This function assumes that `EMPNO` is the Consolidator Manager client id. If the `group_fnc` is not specified, the default grouping is based on subscription parameters.

1.2.7.3 Compatibility and Migration for Shared Maps

If you have been using a version prior to Oracle Database Lite 10g, then you must migrate your existing Mobile Server schema with shared maps, as follows:

1. Run one cycle of MGP.
2. The clients must sync with the server to get the latest changes prepared by the MGP.
3. Stop the Web server and MGP to migrate the server to 10*g*. This automatically sets all the nonupdatable publication items to shared items. If any shared publication items need to use grouping functions or any publication items need to change their sharing attribute, execute custom code that calls the appropriate Consolidator Manager API. See the `SetPublicationItemMetadata` API in Section 1.2.7.2, "Shared Map Usage".
4. The `ShrinkSharedMaps` Consolidator Manager API must be called to set the clients to use shared map data and remove old redundant data from the maps.
5. Start the Web server and MGP.

1.2.8 Use Map Table Partitions to Streamline Users Who Subscribe to a Large Amount of Data

Sync Server database objects called map tables are used to maintain the state for each Mobile client. If there are a large number of clients, and each client subscribes to a large amount of data, the map tables can become very large creating scalability issues. Using the following APIs, map tables can be partitioned by client id, making them more manageable.

The API allows you to create a map table partition, add additional partitions, drop one or all partitions, and merge map table partitions. Map table partitions can be monitored using the `ALL_PARTITIONS` database catalog view.

> **Note:** This form of partitioning is not related to the partition functionality provided by Oracle Server, and is used exclusively by Oracle Database Lite 10*g*.

1.2.8.1 Create a Map Table Partition

Creates a partition for the referenced publication item map table. If there is data in the map table, it is transferred to the partition being created. After the partition has been successfully created, the map table can be truncated to remove redundant data using the SQL command `TRUNCATE TABLE`.

> **Note:** Records removed from the server through a `truncate` command will not be removed from the client unless a complete refresh is triggered. The truncate command is considered a DDL operation. Consequently, the necessary DML triggers do not fire and therefore the operations are not logged for fast refresh.

Syntax

```
public static void partitionMap
   (String pub_item,
    int num_parts,
    String storage,
```

```
    String ind_storage) throws Throwable
```

The parameters of `partitionMap` are listed in Table 1–4.

Table 1–4 The partitionMap Parameters

Parameter	Definition
`pub_item`	The publication item whose map table is being partitioned.
`num_parts`	The number of partitions.
`storage`	A string specifying the storage parameters. This parameter requires the same syntax as the SQL command `CREATE TABLE`. See the *Oracle SQL Reference* for more information.
`ind_storage`	A string specifying the storage parameters for indexes on the partition. This parameter requires the same syntax as the SQL command `CREATE INDEX`. See the *Oracle SQL Reference* for more information.

Example

```
consolidatorManager.partitionMap("P_SAMPLE1", 5, "tablespace mobileadmin",
"initrans 10 pctfree 70");
```

1.2.8.2 Add Map Table Partitions

Adds a partition for the referenced publication item's map table. If there is data in the map table, it is transferred to the partition being created. After the partition has been successfully created, the map table can be truncated to remove redundant data using the SQL command `TRUNCATE TABLE`.

> **Note:** Records removed from the server through a `truncate` command will not be removed from the client unless a complete refresh is triggered. The truncate command is considered a DDL operation. Consequently, the necessary DML triggers do not fire and therefore the operations are not logged for fast refresh.

Syntax

```
public static void addMapPartitions
   ( String pub_item,
    int num_parts,
    String storage,
    String ind_storage) throws Throwable
```

The parameters of `addMapPartitions` are listed in Table 1–5:

Table 1–5 The addMapPartitions Parameters

Parameter	Definition
`pub_item`	The publication item whose map table is being partitioned.
`num_parts`	The number of partitions.
`storage`	A string specifying the storage parameters. This parameter requires the same syntax as the SQL command `CREATE TABLE`. See the *Oracle Database Lite SQL Reference* for more information.

Table 1–5 (Cont.) The addMapPartitions Parameters

Parameter	Definition
ind_storage	A string specifying the storage parameters for indexes on the partition. This parameter requires the same syntax as the SQL command CREATE INDEX. See the *Oracle Database Lite SQL Reference* for more information.

Example

```
consolidatorManager.addMapPartitions("P_SAMEPLE1",5,"tablespace
mobileadmin","initrans 10 pctfree 40");
```

Note: Map Partitions are created only for existing users. New users are placed in the original map table.

1.2.8.3 Drop a Map Table Partition

Drops the named partition. In the following example, the partition parameter is the name of the partition. Partition names must be retrieved by querying the ALL_PARTITIONS table view CV$ALL_PARTITIONS since partitions are named by Data Synchronization.

Syntax

```
public static void dropMapPartition( String partition) throws Throwable
```

Example

```
consolidatorManager.dropMapPartition("MAP101_1");
```

1.2.8.4 Drop All Map Table Partitions

Drops all partitions of the map table for the named publication item.

Syntax

```
public static void dropAllMapPartitions( String pub_item) throws Throwable
```

Example

```
consolidatorManager.dropAllMapPartitions("P_SAMPLE1");
```

1.2.8.5 Merge Map Table Partitions

Merges the data from one partition into another. Partition names must be retrieved by querying the ALL_PARTITIONS table view CV$ALL_PARTITIONS, since partitions are named by Data Synchronization.

Syntax

```
public static void mergeMapPartitions
   ( String from_partition,
    String to_partiton) throws Throwable
```

Example

```
consolidatorManager.mergeMapPartition(""MAP101_1", "MAP101_2");
```

1.2.9 Configuring Back-End Oracle Database to Enhance Synchronization Performance

You can configure the Oracle Database in such a way as to enchance your Mobile Server synchronization performance, as follows:

- Section 1.2.9.1, "Physically Separate Map Tables and Map Indexes"
- Section 1.2.9.2, "Database Parameter Tuning"

1.2.9.1 Physically Separate Map Tables and Map Indexes

During synchronization, map tables are used extensively. Map tables are internal tables, and have table names using the following pattern: `CMP$pub_item_name`. Each map table has four separate indexes. By default, both map table and indexes are created in the default tablespace `SYNCSERVER`.

You can improve performance if you move the map table indexes to a different disk than the map table itself. Create a separate tablespace (for example: `MAPINDEXES`) on a different disk and manually move all indexes. Because the process of moving the indexes requires you to drop and re-create the indexes, you should move the index before many users have synchronized. Otherwise recreating the indexes on the map tables may be very time consuming, as map tables grow with the number of users who have synchronized.

To move the indexes on a map table, do the following:

1. Identify all indexes on the map table (`CMP$pub_item_name`). There are three or four indexes. Move all of them.
2. For each index, record the type of index and column lists.
3. If the index is a primary key index, then remove the primary key constraint on the map table.
4. Drop the index.
5. Recreate the index using the same name, type and column list. Use the storage clause in the create index statement to specify the new tablespace. You may also specify different storage parameters. Refer to the Oracle database documentation for more information on how to create indexes and storage clause parameters.

> **Note:** Repeat step 3 through 5 for all other indexes on the map table.

1.2.9.2 Database Parameter Tuning

Tuning the database for Mobile Server is similar to any Oracle database tuning required for any query intensive applications. Configure the SGA to be as large as possible on your system to maximize the caching capabilities and avoid I/O wherever possible.

Tune your Oracle database with the following database parameters:

- `db_block-buffers`
- `sort_area_size`
- `log_buffers`

Refer to the Oracle database tuning guide for more information on database tuning.

1.2.10 Priority-Based Replication

With priority-based replication, you can limit the number of rows per snapshot by setting the flag **Priority** to 1 (the default is 0).

For example, if you have a snapshot with the following statement:

```
select * from projects where prio_level in (1,2,3,4)
```

With the **Priority** flag set to 0 (the default), all projects with **prio_level** 1,2,3,4 will be replicated.

In a high priority situation, the application can set the flag to 1, which will cause MGP to check for **Restricting Predicate**. A Restricting Predicate is a conditional expression in SQL. The developer can set Restricting Predicate in the **AddPublicationItem()** method, as in the following example:

```
prio_level = 1
```

MGP appends (AND) the expression to the snapshot definitions when composing data for the client. In this case, the high priority statement would be:

```
SELECT * FROM projects where prio_level in (1,2,3,4) AND prio_level = 1;
// a restricting predicate snapshot
```

In this case, only projects with level =1 will be replicated to the client.

This advanced feature is available only through the Consolidator Manager API. It is not available through the Packaging Wizard.

To summarize, there are two steps to enable this feature:

1. Provide a restricting predicate expression in the **AddPublicationItem()** function.
2. Set the PRIORITY flag to 1 in the Mobile Sync API.

> **Note:** You cannot use fast refresh synchronization with high priority.

1.2.11 Caching Publication Item Queries

This feature allows complex publication item queries to be cached. This applies to queries that cannot be optimized by the Oracle query engine. By caching the query in a temporary table, the Sync Server template can join to the snapshot more efficiently.

Storing the data in a temporary table does result in additional overhead to MGP operation, and the decision to use it should only be made after first attempting to optimize the publication item query to perform well inside the Sync Server template. If the query cannot be optimized in this way, the caching method should be used.

The following example is a template used by the MGP during the compose phase to identify client records that are no longer valid, and should be deleted from the client:

```
UPDATE pub_item_map map
SET delete = true
WHERE client = <clientid>
AND NOT EXISTS (SELECT 'EXISTS' FROM
    (<publication item query>) snapshot
     WHERE map.pk = snapshot.pk);
```

In this example, when `<publication item query>` becomes too complex, because it contains multiple nested subqueries, unions, virtual columns, connect by clauses, and other complex functions, the query optimizer is unable to determine an acceptable

plan. This can have a significant impact on performance during the MGP compose phase. Storing the publication item query in a temporary table, using the publication item query caching feature, flattens the query structure and enables the template to effectively join to it.

1.2.11.1 Enabling Publication Item Query Caching

The following API enables publication item query caching.

Syntax

```
public static void enablePublItemQueryCache(String name)
      throws Throwable
```

The parameters for `enablePublItemQueryCache` are listed in Table 1–6:

Table 1–6 The enablePubItemQueryCache Parameters

Parameters	Description
`name`	A string specifying the name of the publication item.

Example

```
consolidatorManager.enablePubItemQueryCache("P_SAMPLE1");
```

If you are using an input string from the input parameter `argv` array, cast it to a `String`, as follows:

```
consolidatorManager.enablePubItemQueryCache( (String) argv[0]);
```

1.2.11.2 Disabling Publication Item Query Caching

The following API disables publication item query caching.

Syntax

```
public static void disablePubItemQueryCache(String name)
      throws Throwable
```

The name parameter for `disablePubItemQueryCache` is listed in Table 1–7:

Table 1–7 The disablePubItemQueryCache Parameters

Parameters	Description
`name`	A string specifying the name of the publication item.

Example

```
consolidatorManager.disablePubItemQueryCache("P_SAMPLE1");
```

1.2.12 Architecture Design of Mobile Server and Oracle Database for Synchronization Performance

It is recommended that you run Mobile Server and the Oracle database on different machines. If possible, use multi-CPU machines for both Mobile Server and the Oracle database. Run the Oracle database should in dedicated server mode; use of the multi-threaded server is not recommended.

1.2.13 Synchronization Disk Needs May Impose on WinCE Platform Available Space

During synchronization, files are created within the Mobile Server directories for synchronization management. This may cause space problems on the WinCE device. To counter space constraints for the storage card on the WinCE platform, you can designate the `Temp` directory for all synchronization temporary files by adding the following entry in the `ALL DATABASES` section in the `POLITE.INI` or `POLITE.TXT` file.

```
TEMP_DIR=\Storage Card\Temp
```

1.2.14 Designing Application Tables and Indexes for Synchronization Performance

Your clients may perform a large number of insert and delete operations on snapshots, and then synchronize their data changes with the Mobile Server. If this is the case, then consider placing the application tables and the indexes on those tables on separate disks.

1.3 Determining Performance of Client SQL Queries With the EXPLAIN PLAN

If you want to access data on the local client Oracle Lite database, then you can use the EXPLAIN PLAN to determine the performance of your SQL query execution on the Oracle Lite database. To execute a SQL statement, Oracle might need to perform several steps. Each of these steps either physically retrieves rows of data from the database or prepares them in some way for the user issuing the statement. The combination of the steps Oracle uses to execute a statement is called an execution plan, which includes an access path for each table that the statement accesses and an ordering of the tables (the join order) with the appropriate join method. The execution plan shows you exactly how Oracle Database Lite executes your SQL statement.

The components of an execution plan include the following:

- An ordering of the tables referenced by the statement.
- An access method for each table mentioned in the statement.
- A join method for tables affected by join operations in the statement.

The EXPLAIN PLAN command stores the execution plan chosen by the Oracle Database Lite optimizer for SELECT, UPDATE, INSERT, and DELETE statement.

You can generate an Explain Plan using either of the following methods:

- The Consperf tool: The Consperf tool generates the following two primary analysis reports:

  1. Timing statistics for publication items
  2. Explain plans for publications

  For a full description of how to use the Consperf utility, see Section 1.2.1 "Analyzing Performance of Publications With the Consperf Utility" in the *Oracle Database Lite Troubleshooting and Tuning Guide*.

- Manually generrated. See the Section 1.11 "Tuning SQL Statement Execution with the EXPLAIN PLAN" in the *Oracle Database Lite SQL Reference* for full details on how to manually create an EXPLAIN PLAN.

1.4 Optimizing Application SQL Queries Against Oracle Lite Database

The following sections provide tips on improving the performance of the application SQL queries against the back-end Oracle database:

- Section 1.4.1, "Optimizing Single-Table Queries"
- Section 1.4.2, "Optimizing Join Queries"
- Section 1.4.3, "Optimizing with Order By and Group By Clauses"
- Section 1.4.4, "Advanced Optimization Techniques for SQL Queries in Oracle Database Lite"

The tip examples use the database schema listed in Table 1–8:

Table 1–8 Database Schema Examples

Tables	Columns	Primary Keys	Foreign Keys
LOCATION	LOC#	LOC#	
	LOC_NAME		
EMP	SS#	SS#	
	NAME		
	JOB_TITLE		
	WORKS_IN		WORKS_IN references DEPT (DEPT#)
DEPT	DEPT#	DEPT#	
	NAME		
	BUDGET		
	LOC		LOC references LOCATION (LOC#)
	MGR		MGR references EMP (SS#)

1.4.1 Optimizing Single-Table Queries

To improve the performance of a query that selects rows of a table based on a specific column value, create an index on that column. For example, the following query performs better if the `NAME` column of the `EMP` table has an index.

```
SELECT *
FROM EMP
WHERE NAME = 'Smith';
```

An index may ruin performance if selecting more than 10% of the rows of the indexing columns is poor. For example, an index on `JOB_TITLE` may not be a good choice even if the query is as follows.

```
SELECT *
FROM EMP
WHERE JOB_TITLE='CLERK'
```

1.4.2 Optimizing Join Queries

The following can improve the performance of a join query (a query with more than one table reference in the FROM clause).

1.4.2.1 Create an Index on the Join Column(s) of the Inner Table

In the following example, the inner table of the join query is DEPT and the join column of DEPT is DEPT#. An index on DEPT.DEPT# improves the performance of the query. In this example, since DEPT# is the primary key of DEPT, an index is implicitly created for it. The optimizer will detect the presence of the index and decide to use DEPT as the inner table. In case there is also an index on EMP.WORKS_IN column the optimizer evaluates the cost of both orders of execution; DEPT followed by EMP (where EMP is the inner table) and EMP followed by DEPT (where DEPT is the inner table) and picks the least expensive execution plan.

```
SELECT e.SS#, e.NAME, d.BUDGET
FROM EMP e, DEPT d
WHERE e.WORKS_IN = DEPT.DEPT#
AND e.JOB_TITLE = 'Manager';
```

1.4.2.2 Bypassing the Query Optimizer

Normally, the optimizer selects the best execution plan, an optimal order of tables to be joined. In case the optimizer is not producing the best execution plan, you can control the order of execution using the HINTS feature. For more information, see the *Oracle Database Lite SQL Reference*.

For example, if you want to select the name of each department along with the name of its manager, you can write the query in one of two ways. In the first example which follows, the hint /*+ordered*/ says to do the join in the order the tables appear in the FROM clause.

```
SELECT /*+ordered*/ d.NAME, e.NAME
FROM DEPT d, EMP e
WHERE d.MGR = e.SS#
```

or:

```
SELECT /*+ordered*/ d.NAME, e.NAME
FROM EMP e, DEPT d
WHERE d.MGR = e.SS#
```

Suppose that there are 10 departments and 1000 employees, and that the inner table in each query has an index on the join column. In the first query, the first table produces 10 qualifying rows (in this case, the whole table). In the second query, the first table produces 1000 qualifying rows. The first query will access the EMP table 10 times and scan the DEPT table once. The second query will scan the EMP table once but will access the DEPT table 1000 times. Therefore the first query performs better. As a rule of thumb, tables should be arranged from smallest effective number of rows to largest effective number of rows. The effective row size of a table in a query is obtained by applying the logical conditions that are resolved entirely on that table.

In another example, consider a query to retrieve the social security numbers and names of employees in a given location, such as New York. According to the sample schema, the query would have three table references in the FROM clause. The three tables could be ordered in six different ways. Although the result is the same regardless of which order you choose, the performance could be quite different.

Suppose the effective row size of the LOCATION table is small, for example `select count(*) from LOCATION where LOC_NAME = 'New York'` is a small set. Based on the above rules, the LOCATION table should be the first table in the FROM clause. There should be an index on LOCATION.LOC_NAME. Since LOCATION must be joined with DEPT, DEPT should be the second table and there should be an index on

the LOC column of DEPT. Similarly, the third table should be EMP and there should be an index on EMP#. You could write this query as:

```
SELECT /*+ordered*/ e.SS#, e.NAME
FROM LOCATION l, DEPT d, EMP e
WHERE l.LOC_NAME = 'New York' AND
l.LOC# = d.LOC AND
d.DEPT# = e.WORKS_IN;
```

1.4.3 Optimizing with Order By and Group By Clauses

Various performance improvements have been made so that SELECT statements run faster and consume less memory cache. Group by and Order by clauses attempt to avoid sorting if a suitable index is available.

1.4.3.1 IN Subquery Conversion

Converts IN subquery to a join when the select list in the subquery is uniquely indexed.

For example, the following IN subquery statement is converted to its corresponding join statement. This assumes that c1 is the primary key of table t2:

```
SELECT c2 FROM t1 WHERE
c2 IN (SELECT c1 FROM t2);
```

becomes:

```
SELECT c2 FROM t1, t2 WHERE t1.c2 = t2.c1;
```

1.4.3.2 ORDER BY Optimization with No GROUP BY

This eliminates the sorting step for an ORDER BY clause in a select statement if ALL of the following conditions are met:

1. All ORDER BY columns are in ascending order or in descending order.
2. Only columns appear in the ORDER BY clause. That is, no expressions are used in the ORDER BY clause.
3. ORDER BY columns are a prefix of some base table index.
4. The estimated cost of accessing by the index is less than the estimated cost of sorting the result set.

1.4.3.3 GROUP BY Optimization with No ORDER BY

This eliminates the sorting step for the grouping operation if GROUP BY columns are the prefix of some base table index.

1.4.3.4 ORDER BY Optimization with GROUP BY

When ORDER BY columns are the prefix of GROUP BY columns, and all columns are sorted in either ascending or in descending order, the sorting step for the query result is eliminated. If GROUP BY columns are the prefix of a base table index, the sorting step in the grouping operation is also eliminated.

1.4.3.5 Cache Subquery Results

If the optimizer determines that the number of rows returned by a subquery is small and the query is non-correlated, then the query result is cached in memory for better performance. For example:

```
select * from t1 where
t1.c1 = (select sum(salary)
from t2 where t2.deptno = 100);
```

1.4.4 Advanced Optimization Techniques for SQL Queries in Oracle Database Lite

> **Note:** This section is provided for those administrators and developers who are already very familiar with optimization techniques for SQL queries. Thus, this material is advanced and not for a beginner.

Unlike procedural languages—such as Java or C—SQL is a declarative language. It states what to do, but does not tell how to do it. This frees developers from writing navigation code to retrieve data. The responsibility of navigation falls on the database management system (DBMS).

The query optimizer—a component of the DBMS—is responsible to come up with an efficient plan to execute the query. Since there are several ways to perform a query, the query optimizer and query execution engine decide how to deliver the result in the quickest time. In a perfect world, the query optimizer will always be right and the database will always be infallible. However, this is not the case. The developer needs to think about the characteristics and peculiarities of the query optimizer. When you do run into performance issues, you can improve the performance as simply as creating some indexes, dropping additional indexes, or re-writing the query. Oracle Database Lite constantly improves the optimizer, so that you do not have to re-write the query.

This section introduces you to the Oracle Database Lite Query Optimizer, briefly covers the architecture of Oracle Lite database, and then provides details of the query compilation and optimization. Lastly, we provide tips on improving query performance. For further information, there are several excellent articles in technical journals that cover SQL query optimization in great technical detail. Some of these journals are listed in the reference section of this document.

1.4.4.1 Oracle Lite Database Application Architecture

The basic database architecture components from the point of view of the application developer are outlined below:

Figure 1–1 Components in applications using ODBC or JDBC

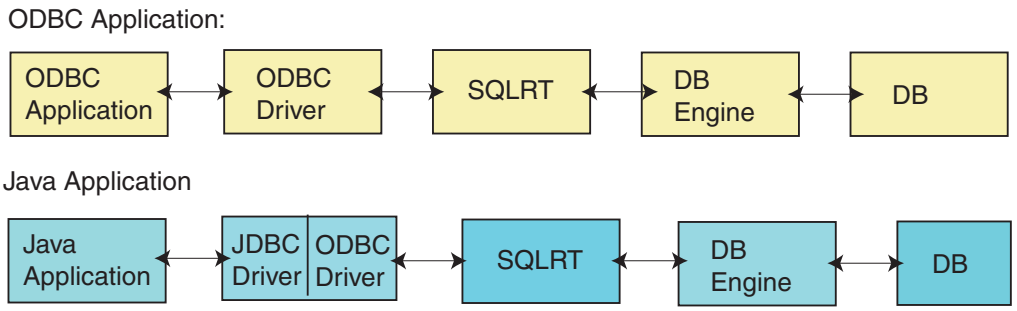


Table 1–9 Architecture Components

Component	Description
<<	Indicates a request and response information flow.
ODBC application	Typically a C or C++ application that issues ODBC API calls.
Java Application	A piece of code written in Java that uses JDBC API to manipulate the database.
ODBC driver	The driver that implements the ODBC API. It calls into the SQL runtime engine (SQLRT).
SQLRT	The SQL Runtime Engine that implements SQL functionality using the capabilities of the underlying database engine.
DB engine and DB	Database engine and Database

1.4.4.1.1 ODBC Application The ODBC application is usually written in C, or C++, or Visual Basic. Third party tools, such as Power Builder, can also generate code that invokes ODBC. The ODBC driver implements ODBC API semantics and uses internal SQLRT APIs to call into SQLRT.

1.4.4.1.2 SQLRT SQLRT, the Oracle Lite SQL engine, is implemented in the `olsql40.dll`. SQLRT implements SQL functionality using the capabilities of underlying database engine. This is covered in some detail in the following sections.

1.4.4.1.3 DB Engine The Oracle Lite database engine implements the object kernel API (also known as OKAPI). The database engine implements an object view of the world; that is, it implements classes, attributes, and iterators.

- Instead of creating a table—which contains a set of columns—you create a class containing a set of attributes.
- Instead of creating a cursor on a table, you create an iterator on a class or a group.

All classes belong to a group, which is a collection of classes.

The DB Engine maintains its own set of Meta catalogs (Meta classes) to store declarative information about classes, attributes, and indexes. For example, see the table below:

Table 1–10 Database Engine Meta Classes

Class	Description
okClass	Information about every class
okAttribute	Information for every attribute in all classes

Table 1–10 (Cont.) Database Engine Meta Classes

Class	Description
okIndex	Information for every index created in the database
okGroup	Information about every group in the database

Note: All object kernel Meta classes belong to the MetaGroup group, which is case sensitive. The DB Engine is responsible to implement the ACID properties of a transaction.

1.4.4.2 Overview of SQL Runtime

The SQL Runtime is responsible for providing a SQL interface to the database. It maps SQL entities to the appropriate object kernel entities and translates all SQL operations into a sequence of basic object kernel primitives. For example, a table is mapped to a class; its table columns are mapped to attributes within the class. The mapping between SQL operations and object kernel APIs is not defined here, as this is not the focus of the document.

Execution of a SQL statement involves the following steps:

1. Compile

   You can compile a SQL statement into an internal representation that is easy and efficient to execute. A SQL statement can be one of the following:

   - DDL (data definition language): An example of a DDL statement is "`CREATE INDEX emp_last_name ON employee (last_name, first_name)`".
   - DML (data manipulation) statement: Examples of DML statements are `SELECT`, `INSERT`, `UPDATE`, `DELETE` and `COMMIT` statements.

2. Bind

   A SQL statement may contain markers (such as "`?`"), which are used as placeholders for parameters that can be supplied before execution of the statement. Binding sets the value for each marker in the SQL statement.

3. Execute

   This is when a previously compiled statement is executed. Execution involves interpretation of the internal representation of the SQL statement and making all calls into the database engine to achieve the desired result. The following are examples of what the execution means for particular statements:

   - For an index creation statement, the index is created.
   - For an `INSERT` statement, the row is inserted into the table. In the object terminology, a new object is created.
   - For a `SELECT` statement, the statement is executed, where a row is available for retrieval. The execution of a `SELECT` statement produces a result set, which is a set of rows. It is not necessary that all rows be materialized. However, for a `READ COMMITED` isolation level transaction, all rows are materialized at this step.

4. Fetch

   This step is required for a `SELECT` statement. Every fetch call returns one row to the caller.

5. Close

   Close the result set created in the execute step. Any remaining rows, if any, are discarded and any resources tied to the processing of the statement are freed up.

1.4.4.2.1 Compilation Compilation is somewhat like translating a JAVA program into byte code. In SQLRT, we translate a SQL statement into an internal data structure called a tree. The following are the steps SQLRT goes through to generate the execution tree, which determines the best method to execute that statement:

1. Parsing: The input statement is scanned and is converted into an abstract syntax tree. Grammatically incorrect statements are flagged and any syntax error is returned.
2. Normalization: The tree is walked, normalized and decorated. Transformation is carried out and semantic information is added to the tree nodes. Any tautologies are evaluated.

   For Example ((1 = 0) AND C1 > 5 OR C2 = 7) is evaluated to (C2 = 7). Any semantic error is caught during the tree traversal, such as Data type mismatches in expressions or SQL operations, references to non existing tables, or columns, unsupported SQL operations, and so on.
3. View expansion: Any references to views are expanded in line and the view tree is walked.
4. View Optimization: If possible, the view expansions are collapsed into the main queries. For example, the statement "SELECT * FROM v1,v2 where v1.c1=v2.c2" is resolved to a query on the base tables in v1 and v2. The transformation takes place on the query tree. This merging may not be possible. For example, if a view selects aggregate functions (COUNT, AVG, and so on.) or contains UNION or MINUS operators, it cannot be collapsed.
5. Subquery optimization: You can re-write the query to eliminate the subquery. This technique is called subquery un-nesting. The tables and filter conditions in the where clause are moved to the parent query block. This is possible only when the subquery does not contains any aggregates, `UNION`, or `MINUS` operations and SQLRT can make sure that the subquery does not return any duplicate rows.
6. Transitive Closure of Predicates: Predicates are analyzed and extra inferences are added to the `WHERE` clause, which helps the optimizer in selecting the best execution plan.
7. Predicate Push: The predicates are pushed down from top to bottom, which helps the queries on top of views. When a view contains any `UNION`, `MINUS` and `GROUP BY` clauses, it helps to push the filtering condition to the source of data or base tables.
8. Execution Plan Generation: The query is now analyzed to generate the best execution plan, which is based on a cost-based approach.
9. Query Execution: The execution plan generated is used to execute the query.

1.4.4.2.2 Query Tree Transformations or Query Re-write Examples These are examples of query tree transformations or query re-writes.

- View Optimization Example for View Replacement or Merging
- View Expansion and Predicate Push
- Subquery Transformation

View Optimization Example for View Replacement or Merging

Consider the following statements:

1. `SQL> CREATE VIEW v_dept_emp AS SELECT emp.*, dept.dname, loc FROM emp, dept WHERE emp.deptno=dept.deptno;`
2. `SELECT * FROM v_dept_emp WHERE loc = 'DALLAS';`

The query tree transformation process substitutes the definition of view `v_dept_emp` into the select query and collapses the query into single level query. The query then becomes as follows:

```
SELECT  emp.*, dept.dname, dept.loc FROM emp, dept WHERE emp.deptno=dept.deptno
and loc = 'DALLAS'
```

> **Note:** The final query does not refer to the view.

View Expansion and Predicate Push

Consider the following example:

```
SQL> CREATE VIEW v_sal_expense (dno, name, total_sal) AS SELECT dept.deptno,
dept.dname, sum(sal)  FROM emp, dept WHERE emp.deptno=dept.deptno group by
dept.deptno, dname;

SELECT  * FROM v_sal_expense WHERE  total_sal > 10000;
```

Since the query involves aggregation, it cannot be collapsed into the main query and the query after re-write is as follows:

```
SELECT  * FROM (
   SELECT dept.deptno, dept.dname, sum(sal)  total_sal
   FROM emp, dept
   WHERE emp.deptno=dept.deptno
   group by dept.deptno, dname)  temp_view
WHERE temp_view.total_sal > 10000;
```

> **Note:** The predicate `total_sal > 10000` was not pushed into the inner query block as `total_sal` refers to an aggregate `sum(sal)` column in the view definition.

Consider the following query on the same view:

```
SELECT  * FROM v_sal_expense WHERE  dno = 10;
```

The query after the re-write is as follows:

```
SELECT  * FROM (
   SELECT dept.deptno, dept.dname, sum(sal)  total_sal
   FROM emp, dept
   WHERE emp.deptno=dept.deptno and
   dept.deptno = 10
   group by dept.deptno, dname)  temp_view
WHERE dno=10;
```

The predicate `dept.deptno = 10` is pushed down into the nested view expansion, which demonstrates the Predicate Push optimization. The aggregation is performed for the department number 10 only; therefore, this query performs better.

Subquery Transformation

Consider the following query:

```
SELECT * FROM emp WHERE emp.deptno IN (SELECT deptno
          FROM dept WHERE loc = 'DALLAS');
```

Since the subquery selects a unique key column (`deptno`), it can be converted into a join query. In general, a join query provides more flexibility in optimization and performs better. This query could be transformed as follows:

```
SELECT emp.* FROM emp, dept
   WHERE emp.deptno = dept.deptno AND dept.loc = 'DALLAS';
```

Note: The above subquery is a non-correlated subquery; that is, the subquery does not make a reference to columns from the tables in the outer query. For a non-correlated query, Oracle Database Lite does not always transform it to a join query. Instead, sometimes it decides to cache the query result into memory and perform the IN operation at run-time. A correlated subquery, if it meets the correctness requirements, is always transformed into a join query, as follows:

```
SELECT * FROM emp WHERE emp.deptno IN (SELECT deptno FROM dept
WHERE loc = 'DALLAS' AND emp.deptno = dept.depno);
```

Which is transformed into the following:

```
SELECT emp.* FROM emp, dept WHERE emp.deptno = dept.deptno AND
dept.loc = 'DALLAS' AND emp.deptno = dept.depno;
```

1.4.4.3 Execution Plan Generation

Execution plan generation is the last step of query compilation. It is the responsibility of the query optimizer to find the least expensive plan. It generates all plausible execution plans and picks the least expensive plan. As the number of tables in a query increases, the cost of evaluating all possible orders of execution increases exponentially. The optimizer uses its own heuristics to reduce the search space. The query optimizer considers only I/O costs for comparing the different execution plans. It does not consider the CPU time used to perform different operations. The I/O cost is computed based on the statistical information available to it; therefore, the quality of cost estimation depends upon the quality of statistics available.

- Section 1.4.4.3.1, "Statistics"
- Section 1.4.4.3.2, "Access Methods"
- Section 1.4.4.3.3, "Single Table I/O Cost"
- Section 1.4.4.3.4, "Join Query Optimization"

1.4.4.3.1 Statistics The Oracle Database Lite engine maintains the following statistics at all times. You do not have to run a separate command to update the statistics.

Table 1–11 Oracle Database Lite Engine Statistic Parameters

Parameter	Description
npg	Number of data pages allocated to each table.
nrows	Number of rows in the table.
ndk	For each index, number of distinct keys.

Table 1–11 (Cont.) Oracle Database Lite Engine Statistic Parameters

Parameter	Description
nrangeSize	For each index, OKAPI supports an API to estimate the number of rows selected for a given range of key values.
nMaxKey	Maximum value of a key (an OKAPI call is used to estimate it).
nMinKey	Minimum value of a key (an OKAPI call is used to estimate it).

Selectivity Factor

To estimate I/O cost, the optimizer estimates the number of pages that will be read or written to satisfy the query. It evaluates the disk I/O costs for different execution plans before selecting the best one. It assigns a selectivity factor to each predicate (also called a factor in boolean algebra), which is defined as an expected fraction of rows and satisfies the predicate. That is, the selectivity factor is defined as follows:

```
Selectivity factor = (expected-number-rows)/(total-number-of-rows)
```

The current values of the selectivity factor are as follows:

Note: The values are subject to change without any notice.

Table 1–12 Selectivity Factor Values

Condition	Example	Default	With Index
Equality	Name = 'Smith'	1/5	1/ndk
Range	C1 > 5	1/2	Pretty good estimate
Between	C1 between (4,10) orC1 > 4 and C1 < 10	1/3	Pretty good estimate
Is Null	C1 is NULL	1/10	1/10
Like	Name Like 'Sm%'	1/3	Estimate*

Like: A like predicate is transformed into a range and like. The range predicate is then appropriately optimized. For example, Name like 'S%' is converted into Name like 'S%' AND Name >= 'S' AND Name < 'T'. Now the range ('S', 'T') for Name can be used to calculate the selectivity.

Not Equal: Selectivity factor for not equal is as follows: (1-Selectivity factor for the equal operator).

Caveat With Bind Variables

When bind variables are present, then the selectivity factor for "range", "between" and "like" cannot be correctly estimated and the default selectivity factor is used.

1.4.4.3.2 Access Methods An important component of an execution plan is the "access method" used for each table involved in the execution plan. The Oracle Lite database engine supports the following access methods:

1. A Full table scan: All pages of the table are searched. Therefore, the cost of retrieval is equal to npg (the number pages) in the table.
2. Index access method: A key value or key range—such as, price between (10,15)—is used to retrieve the qualifying rows. The key or key range is used to find the row-ids of the matching rows. Each row-id uniquely identifies the location of the

row in the database. The rows are fetched in increasing or decreasing order of the key, which is useful when optimizing queries containing order by or group by clauses.

Cost of Access Methods

The I/O Cost can be computed in terms of the following parameters:

Table 1–13 I/O Access Method Cost

Parameter	Description
npg	Number of data pages.
nrows	Number of rows in the table.
h	Height of the index. It is also called depth of an index tree.
nlf	Number of leaf pages in an index tree.
sf	Expected Fraction of the number of rows selected. It is between 0 and 1.

Since the values for "h" and "nlf" are not available, its values are improvised based on nrows and estimated key size.

Cost of a Full Table Scan

The cost of a full table scan is the number of data pages, as follows: `Cost = npg`.

Cost of an Index Scan

The cost of an index scan is approximated to be as follows:

```
Cost = the number of index pages read + the number of data pages read
```

Where: `number of index pages read = (h-2) + ceil(nlf * sf)`. The value for `h` is calculated based on the estimated key size and number of rows.

It is assumed that the root of index tree is always in memory. Thus, the cost of reading the root page is ignored. Assuming that only a small number of rows are selected by the index access method, we approximate the number of leaf pages read to be one. This is performed sine we do not have information about nlf. Even for a range scan, we approximate it to be one.

For a primary key index or for an index with ndk/nrows close to one, we assume the data to be clustered on the key column values and we estimate the number of data pages read as follows:

```
Number of data pages read = ceil(sf * npg)
```

If the index is not a primary key index, then there is a good chance that the consecutive key values will not be on the same data page. Each new row access can potentially read a new page. The number of data pages read will be in between `sf *npg` and `sf * nrows`. We use the following formula as an approximation to actual number of data pages read:

```
Number of data pages read = ceil (sf * sqrt (npg, nrows))
```

Therefore, the cost of index access is as follows:

- For a clustered index, the cost is = `(h-1) + ceil(sf * npg)`.
- For a non-clustered index, the cost is = `(h-1) + ceil(sf*sqrt(npg,nrows))`.

1.4.4.3.3 Single Table I/O Cost To find the optimal execution plan for a single table query, the costs for each possible access methods are evaluated and the least expensive access method is picked. For example:

```
SELECT * FROM T1 where C1 between 1 and 5 AND C2 > 5 and C2 < 100;
```

Assuming that the indexes exist on C1 and C2, then the optimizer estimates the selectivity for predicates "C1 between 1 and 5" and "C2 > 5 and C2 < 100". It then computes the I/O cost for retrieving the rows using a full table scan, an index scan on C1, and an index scan on C2. The access method that produces the least amount of I/O is chosen.

Interesting Order Optimization

For a single table query that contains "order by" or "group by" clause, the interesting order optimization technique is used to influence which access method is chosen. The result set size and sorting cost are estimated. Sorting can be avoided, if an index is available that can return the rows in the right order. If it is less costly to use an execution plan that does not involve any sort, then it will be chosen. The size of the result set is given by the following:

```
Number of rows in the result set =
     nrows * min(selectivity values for each predicate in the where clause)
```

1.4.4.3.4 Join Query Optimization The join query optimization involves evaluation of a large number of query execution plans. The number of possible plans increases exponentially with the number of tables. The following query illustrates this:

```
SELECT e.empno, e.ENAME, d.dname
FROM EMP e, DEPT d
WHERE e.deptno = DEPT.DEPTNO
AND e.JOB = 'MANAGER'
AND e.sal > 2000;
```

Here both possible orders of (`EMP`, `DEPT`) or (`DEPT`, `EMP`) exist for the execution. If EMP is chosen as the driving table, then the rows qualifying (`EMP.JOB_TITLE = 'Manager' AND EMP.sal > 5000`) are retrieved one by one from the `EMP` table. The optimizer considers the three possible access methods for `EMP` table, as follows:

1. Sequential scan of `EMP` table.
2. Index access using index on `EMP.JOB_TITLE` if one exists.
3. Index access using index on `EMP.SAL` if one exists.

The optimizer picks the method that produces least amount of I/O. Based on the selectivity factor assigned to each predicate, it estimates the number of rows selected for the `EMP` table. Then, it estimates the cost of retrieving a set of matching rows for each outer row in the `EMP` table. The total cost of execution using this order is as follows:

$$Cost = npg_{emp} + est\_row_{emp} * (cost\_per\_row\_dept)$$

Where:

Table 1–14

Parameter	Description
est_rowemp	Estimated number of rows fetched from EMP table
cost_per_row_dept	Cost of index access into DEPT to retrieve matching department rows for each row fetched from EMP

The same calculation is repeated for the order DEPT, EMP. Whichever order produces the lowest cost is chosen. As the number of predicates and tables increase the cost of computing, the different possibilities grow exponentially. To reduce the compilation time, Oracle Lite uses aggressive heuristics to prune the search space, thereby sometimes landing into a sub-optimal execution plan. Also, unreliable statistics values, skewed data, and unavailability of selectivity factors for non-index columns can contribute to sub-optimal execution plan generation.

The following are the main tasks performed during a join query optimization:

1. The optimizer isolates local predicates (the predicates on a single table) from join predicates. In addition, the optimizer estimates the effective table sizes by the applying the selectivity factor of local predicates to the table. Local predicates are predicates that refer to columns from one table only. Whenever an index is available, the calculation of selectivity factor is fairly accurate. Oracle Lite assumes that the data is uniformly distributed; however, when the data is skewed, the estimate can go wrong and the execution plan chosen may not be optimal. When an index is not available, it uses default selectivity for computation.
2. A driving table—the table with the smallest effective cardinality—is picked first. Its optimal access method is picked. The table is put in the set of "outer" tables.
3. The query is examined to discover which possible tables can be joined to the tables in the current outer tables. The cross product is not considered. The I/O cost is estimated for all possible joins. The least costly join is chosen and is added, along with the chosen table, to the outer table set. The step is repeated until it has selected all tables in the query. By the end, it has computed the execution order and access methods for each table in the query.
4. The optimizer saves the current execution plan and picks a new driving table, whose effective cardinality is the second lowest. It repeats step 3 and selects the least expensive execution plan of the two plans. Again, it repeats step 3 with the third, fourth and fifth smallest table—always keeping a copy of the current least expensive execution plan.
5. The optimizer creates hints for when to create an index for intermediate results of a view. This is useful when joining a view that is not collapsed to another table or view.
6. When two tables are outer joined, the master table has to be scanned before the slave table (the table whose column has "+" in the joining column).

Interesting Order Optimization

An interesting order optimization eliminates the final sorting step for queries containing order by or group by clauses. If a suitable index exists that can eliminate the sorting step, then the cost is estimated the following ways:

1. Sorting + the best execution plan.
2. Pick a drive table that has columns from order by or group by clause, such that an index can be used to retrieve the data in the right order. Estimate the execution plan cost.

The least expensive plan is then chosen.

1.4.4.4 Query Execution Engine

The SQL Runtime engine relies on the database engine to implement simple data filtering, access methods, and sorting. A single table query execution of a query involves the following steps:

1. Decide if a temporary table is necessary to store the result. For a `READ COMMITED` isolation level transaction, a temporary table is required to store the result. While the result is being materialized, all other transactions are blocked from committing to preserve the read committed semantics. A temporary table is necessary when sorting is required. The DBE can only sort full tables.
2. Create the iterator on the table. Push the maximum number filter conditions to the database engine. This way, the smaller result set is returned to `SQLRT`.
3. If there are any complex filters that cannot be evaluated by DBE, evaluate them now and reject any rows that do not qualify. Examples of complex filters are SQL functions and subquery operators, such as `UPPER(NAME) = 'SMITH'`, or `zipcode IN (SELECT zipcode from TX where ….)`.
4. If the temporary table is created, then store all qualifying rows into this table. Once all rows are inserted into the temporary table, then the result is returned from this table.

1.4.4.4.1 Join Query Execution The `SQLRT` implements the join operation by executing the query in a nested loop fashion. The optimizer has already picked the optimal order of tables. The execution begins with the first (outer most) table in the list. An iterator is created on this table. A qualifying row is retrieved. The next table is picked from the list and a new iterator is created using the qualifying values from the row already fetched. A new qualifying row is retrieved from the second table. If there are more tables in the list, then the process continues until you reach end of the list. This provides the first matching row for the SELECT statement. Find the next matching row from the last table. If you do not find any qualifying rows, then return to the previous table in the list and repeat the process. Every time you advance to the next table in the list, you create a new iterator. Every time you do not find any more matching rows on a table, then close the iterator and return to the previous table in the list. If you exhaust all rows in the outer most table, then you have found all rows. The execution is analogous to nested loops execution in a programming language, which is why it is called a nested loop join.

1.4.4.4.2 Nested View Execution Oracle Database Lite does not distinguish between dynamic views (the query block in the `FROM` clause) or a view table being used in the `FROM` clause. Both are processed in the same way. If a nested view cannot be merged with the containing query and it is not the first to be picked in the execution order, then `SQLRT` materializes the view into a temporary table and creates a temporary index on the joining column(s). The index is used for joining outer tables with the view. Since the index is created at runtime, the optimizer does not have access to selectivity factors for view columns. The order chosen by the optimizer is based on default selectivity factors and estimated number of rows in the view.

1.4.4.5 Optimization Tips

This section provides guidelines to debug performance problems and helps you design better queries for the Oracle Lite database. Query optimization refers to the techniques used to find best execution plan.

- Section 1.4.4.5.1, "Index Access Method"
- Section 1.4.4.5.2, "Identifying The Bottleneck"
- Section 1.4.4.5.3, "Single Table Query Blocks"
- Section 1.4.4.5.4, "Query Blocks Containing Multiple Tables"
- Section 1.4.4.5.5, "Known Limitations"

1.4.4.5.1 Index Access Method An index access method can be used for equality, as well as range predicates. A predicate has to be one of the following forms in order for it to be considered for index access:

- column_1 = value1
- column_1 rel-op value
- column_1 = value1 AND column_2 = value2 AND …
- column_1 = value1 AND column_2 = value2 AND … column_n rel-op value-n

Where:

- rel-op—One of "=". ">", "<", ">=", <="
- column_n—Prefix columns of an index key. The value is an expression that can be evaluated to a constant value. For example, UPPER(name) = 'TOM' cannot be used with an index access method, UPPER(name) is not a column name, but an expression on the column name. Whereas name = UPPER('TOM') can be used as an index predicate; the right hand side is a constant expression.

Note: You should not create indexes on a column that has multiple duplicate values; that is, the ratio of `nrows/ndk` to `ndk` is large.

1.4.4.5.2 Identifying The Bottleneck The largest problem of solving a query optimization problem is identifying the performance bottlenecks. Where is the CPU spending time? A typical customer query contains multiple tables, predicates, and subqueries. It may also contain outer joins and `UNION` operations. We recommend the following steps to debug the problem:

1. Replace all synonyms with base objects. Expand all views by corresponding view definitions. Imagine how `SQLRT` processes the query and carries out all possible transformations. Identify all query blocks, where each query block contains one `SELECT` statement.
2. Experiment with different query blocks one by one and find the slowest performing query block.
3. Optimize the problematic query block by examining the indexes already existing on columns involved in the query block. Determine if creating new indexes or dropping some indexes improves the performance. Check the order of tables selected by the optimizer (See the "Tools" section). Can it be improved if the query is executed using a different execution order? You can use a HINT to force the execution order of tables.
4. Once the bottleneck is resolved, repeat the process for the next bottleneck.

Tools

In Oracle Database Lite, you can dump query execution plan information by enabling `SQL TRACING`, which is enabled by including the following line in the `polite.ini` configuration file.

```
OLITE_SQL_TRACE= yes
```

This creates an `oldb_trc.txt` file in the current working directory of the process accessing the database. If the file already exists, then it opens the file for appending the dump information. The dump contains the following basic information:

1. Text of the SQL statement and every views in the SQL statement.

2. The time taken to compile the query.
3. The value of each bind variables.
4. Order of joining the tables.
5. Temporary tables created.
6. Access method used for each table. For an index access method, it prints the index name and index number. If the index name is blank, then you can use `idxinfo.exe` to discover the index information.

1.4.4.5.3 Single Table Query Blocks For a single table query, the query optimizer does not select the best available access method. However, it does collect statistics for all available indexes. The job for selecting the best index is left to the DBEngine, which uses a rule-based approach to select the appropriate index. Ensure that the index is available for the highest selective columns, as shown in the following example:

```
SELECT * FROM EMP WHERE NAME = 'Smith' and  EmpNo  between 1 and 1000;
```

Assuming that the total number of employees is a few thousand, then we would expect the predicate `NAME = 'Smith'` to return fewer rows than the predicate `EmpNO` between 1 and 1000. Therefore, we should create an index on the `NAME` column.

> **Note:** Since the DBEngine is following a rule-based approach and EMPNO is a primary key column, it may not select the index on the NAME column.

1.4.4.5.4 Query Blocks Containing Multiple Tables Due to limitations of availability of statistics, and inherent assumptions made about the distribution of data, the execution plan chosen is not always optimal. Also, when suitable indexes are not present, the Oracle Lite Database Engine uses a sequential scan, as opposed to an index access method. To illustrate the importance of the index, see the following query:

```
SELECT e.empno, e.ENAME, d.dname FROM EMP e, DEPT d
WHERE e.deptno = DEPT.DEPTNO AND e.JOB = 'MANAGER' AND e.sal > 2000;
```

$cost = npg_{emp} + est\_row_{emp} * (cost\_per\_row\_dept)$

Let us assume that EMP has 1000 rows with 50 rows per page; that is, the npgemp = 20. Let us assume that the est_row_{emp} is 50, npg_{dept} = 10 and the cost of the index access into the department is 2. The cost calculation is tabulated, as follows:

Table 1–15 Cost Calculation Tabulation

npg_{emp}	est_row_{emp}	npg_{dept}	Access Method dept	cost_per_row_dept	cost
20	50	10	Sequential	10	520 pages
20	50	10	Indexed	2	120 pages

The cost of execution changes dramatically when an index is present. Therefore, the biggest challenge to improve the performance of a query in an Oracle Lite database is as follows:

1. Find the right set of indexes.
2. Optimal order for execution of tables.

There is no easy answer to the above tasks. It requires a deep understanding of the query that you are writing. The first action is to figure out the driving table, which is usually a table with many conditions involving constant values. For example, in the above table, it is most likely the EMP table. On the other hand, if there are only couple of rows in the DEPT table, then the DEPT table is a good candidate for the driving table. Once you select a driving table, the next task is to figure out the possible tables that can be joined to this table. What indexes will help in joining the current result set to the new table? Try joining these two tables and test if the time you receive makes sense. Now, add the third table and repeat the process. To force a specific join order, you can use the HINT clause supported by the Oracle Lite Database. Refer to the *Oracle Database Lite SQL Reference* for more information.

1.4.4.5.5 Known Limitations **1.**In the process of finding the maximum and minimum values for an index key, the optimizer can spend too much time if there are large number of duplicates values near maximum and minimum values.

2. Sorting cost calculation is arbitrary.

3. In the presence of host variables, the selectivity for a range or like predicate cannot be correctly estimated.

1.4.4.6 Glossary

- API - Application Programming Interface
- ACID - ACID properties refer to atomicity, consistency, isolation, and durability
- A Correlated Subquery - A subquery that references columns from tables that are not present in its own "from" clause.
- Cross Product - When you join two tables without any joining condition, you produce a cross product. The cross product of a table containing m rows with another table containing n rows produces (m x n) rows.
- OKAPI -- Object Kernel Application Program Interface is implemented by the Oracle Lite Database Engine, which you can use to program your database application.
- Predicate – A boolean condition that evaluates to "true", "false" or unknown. Examples are: (Emp.Salary > 50000), (Dept.DepNo = 10 AND Emp.HireDate > '17-Nov-2001')
- SQLRT – Oracle Lite SQL Runtime Engine that is responsible for implementing SQL functionality on top of Oracle Lite database engine.

1.4.4.7 References

1. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A.,Price T.G. Access Path Selection in a Relational Database System. In Readings in Database Systems. Morgan Kaufman. This is a classical paper and must read for any one who wants to learn about query optimization.

2. Surajit Chaudhuri, An Overview of Query Optimization in Relational Systems, Microsoft Research

1.5 Maximizing JVM Performance By Managing Java Memory

You can maximize your JVM performance by modifying the amount of memory used by the three areas of Java memory. This is fully described in Section 2.5.1, "Troubleshooting An Out of Memory Error".

2

Troubleshooting

This document contains frequently asked questions for troubleshooting the Mobile Server. Topics include:

2.1 Troubleshooting Synchronization

The following sections describe how to troubleshoot the synchronization process or what to do in the event of certain synchronization scenarios:

2.1.1 Synchronization Errors and Conflicts

Consult the following sections for details on how to resolve any synchronization errors or conflicts:

2.1.1.1 General Synchronization Errors and Conflicts

With the Mobile Server, you can have the following errors when synchronizing: nullity violations, foreign key constraint violations, or the client updates a row at the same time that the server deletes it.

The Mobile Server does not automatically resolve synchronization errors. Instead, the Mobile Server rolls back the corresponding transactions, and moves the transaction operations into the Mobile Server error queue. It is up to the administrator to view the error queue and determine if the correct action occurred. If not, the administrator must correct and re-execute the transaction. If it did execute correctly, then purge the transaction from the error queue.

A Mobile Server synchronization conflict occurs if:

- Nullity violations.
- Foreign key constraint violations.
- The client and the server update the same row.
- The client and server create rows with the same primary key values.
- The client deletes the same row that the server updates.
- The client updates a row at the same time that the server deletes it.

See Section 3.11, "Resolving Conflict Resolution with Winning Rules" in the *Oracle Database Lite Developer's Guide* for more information on conflict resolution techniques.

2.1.1.2 Synchronization Error if Client Device Clock is Inaccurate

The client device clock must be accurate within the timezone set on the device before attempting to synchronize. An inaccurate time may result in the following exception during synchronization: `CNS: 9026 "Wrong username or password. Please enter correct value and reSync."`

2.1.2 Problems When Synchronizing Large Number of Rows

When you synchronize a large number of rows, you may want to set the `AUTO_COMMIT_COUNT` parameter in the `POLITE.INI` file to a smaller number. The smaller the number is, the more often a commit occurs. If you do not set the `AUTO_COMMIT_COUNT` parameter to a smaller number, you may receive an `OutOfMemory` error.

See Section G.3.2.5, "AUTO_COMMIT_COUNT" in the *Oracle Database Lite Administration and Deployment Guide* for information on this parameter.

2.1.3 First Synchronization Causes Browser to Timeout

The duration of the first synchronization process, between the client and the server may take a very long time (For example, upwards of 45 minutes), causing the Microsoft Internet Explorer browser to time out.

The solution is available only for the Microsoft Internet Explorer. For the Mobile Client for Web-to-Go, change the `ReceiveTimeout` value for a particular registry key on Windows 32. The following paragraphs provide the location of this parameter and specifies how to change its value.

1. If you want to change the `ReceiveTimeout` value (that is, the number of milliseconds that the browser will wait to receive the data from the server), uncomment the following two lines in the **REGISTRY** section of the file **setup.ini**.

This file is downloaded to the client from the server when the Mobile Client for Web-to-Go is first installed on the client machine.

```
#KEY: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\
       Internet Settings
#VALUEDWORD: ReceiveTimeout = 40000000
```

2. To uncomment the two lines, remove the hash marks in front of the `KEY` and `VALUEDWORD` statements and then change the `ReceiveTimeout` value to the desired value in milliseconds. In the example given below, the timeout value is `40000 seconds`.

```
wsh -o mobileadmin/manager@webtogo.world
cd setup
edit setup.ini
```

3. When you modify the file **setup.ini** in the Mobile Server Repository, the next time that the file is downloaded to the Mobile Client, these two lines will be uncommented.

2.1.4 Situations Where the Client is Out of Sync that Triggers a Complete Refresh

When a client is out of sync with the server, any outstanding uploaded transaction from the client is placed in the error queue and a complete refresh is triggered to re-initialize the client data with what is currently on the server.

The following are a list of the situations—ordered from most to least likely—that can trigger a complete refresh for the client:

- Dropping and then republishing the application.
- Synchronizing by the same Mobile user from multiple devices on the same platform or from different platforms when the publications are not platform-specific.
- Receiving unexpected server apply phase conditions—such as constraint violations, unresolved conflicts, other database exceptions.
- Modifying the application—such as changing subsetting parameters, or adding or altering publication items.
- Requesting a force refresh from either the server admin or client.
- Two separate applications using the same backend store.
- Unexpected client apply conditions—such as deleting, moving or restoring the Oracle Lite database, database corruption, memory corruption, and other general system failures.
- Loss of transaction integrity between server and client. The server fails post processing after completing a download and disconnecting from the client.
- Data transport corruptions.

2.1.5 The "Inconsistent Datatypes" SQLException Received If Order is Not Correct in Query

If you are creating a fast refresh publication item on a table with a composite primary key, the snapshot query should list the primary key columns in the order that they are present in the table definition. This automatically happens during the column selection when MDW is used or when a `SELECT *` query is used. Note that the order of the

primary key columns in the table definition may be different from those in the primary key constraint definition.

The following example demonstrates what is valid or invalid given the table definition for TAB1:

```
CREATE TABLE TAB1(
 ID1 NUMBER(10) NOT NULL,
 ID2 NUMBER NOT NULL,
 COL1 VARCHAR2(200),
 COL2 VARCHAR2(200),
 ID3 NUMBER(4) NOT NULL);
ALTER TABLE TAB1 ADD CONSTRAINT TAB1_PK PRIMARY KEY (ID3, ID2, ID1);
```

The following are valid snapshot queries:

```
SELECT * FROM TAB1
SELECT ID1,ID2,ID3,COL1,COL2 FROM TAB1
SELECT ID1,ID2,COL1,COL2,ID3 FROM TAB1
```

The following are invalid snapshot queries:

```
SELECT ID3,ID2,ID1,COL1,COL2 FROM TAB1
SELECT ID3,ID2,COL1,COL2 ID1 FROM TAB1
```

Define the table columns where the primary key columns appear before other columns. The order of the primary key columns in the table definition order must match the constraint definition in the snapshot query.

2.1.6 MGP Compose Postponed Due to Transaction in the In-Queue

If the user synchronized and uploaded some more changes after the last apply cycle for a particular user; by default, the MGP must first apply thse changes before it can compose. If this keeps happening, the compose could be postponed beyond what you would like. By default, the MGP tries to avoid a compose phase postponed due to a transaction in the in-queue by performing an apply for any unprocessed in-queue data before doing a new compose. However, if the MGP compose is postponed due to a transaction in the in-queue, you can modify the following parameters to avoid the error:

- `SKIP_INQ_CHK_BFR_COMPOSE`: By default, this parameter is set to NO. Setting this parameter to YES, then a compose is performed for a client even if there is unprocessed data in the in-queue.
- `DO_APPLY_BFR_COMPOSE`: By default, this parameter is set to YES. If set to YES, the unprocessed data in the in-queue is applied before a client compose. This parameter takes effect only if `SKIP_INQ_CHK_BFR_COMPOSE` is set to NO.

For most situations, preserving the default values for these two parameters avoids the occurrence of the MGP Compose postponed error.

2.1.7 Avoiding the Server Busy Warning

The Server Busy warning can be thrown for one of the following reasons:

- When the MGP is processing apply/compose for that user.—To avoid MGP contention with sychronization, MGP should be scheduled to run when few clients are synchronizing. Alternatively, you could use queue-based synchronization, which does not use the MGP at all; thus, avoiding MGP contention with synchronizaiton.

- If a previous synchronization was interrupted for that user and Oracle Database Lite rolls back the transaction—If the Server Busy warning is a result of a long rollback, then Oracle Database recommended tuning steps for rollback operation may reduce the Server Busy state for the client.

2.1.8 Enabling Online Web-to-Go Applications on the Mobile Server Host

When you publish you application, you can immediately initialize a synchronization from the client. However, if you want to use the application on the same host as the Mobile Server, you will receive an error.

The OC4J layer under the Mobile Server does not support auto deployment of the Web application in the Online Mode. If the user wants to access the Web application in the online mode immediately after publishing, then the administrator must restart the Mobile Server.

If you only have the random Web application that you wish to access in this manner, just restart the Mobile Server. However, if you want to have multiple applications to be automatically deployed, you can enable the auto deployment option. This is not recommended as it has an adverse effect on the performance of the Mobile Server.

To enable the auto deployment in the Mobile Server, perform the following:

1. Open the *<J2EE_HOME>*`\mobileserver\config\server.xml` file.
2. Modify the `check-for-updates` parameter from `adminClientOnly` to `all`.

> **Note:** Modifying this parameter enables the Mobile Server to recognize any newly published applications on the host of the Mobile Server. However, it also degrades performance.

3. Save the `server.xml` file.
4. Restart the Mobile Server.

2.2 Troubleshooting the Mobile Server

The following sections detail how to troubleshoot the Mobile Server and its repository:

- Section 2.2.1, "Running the Mobile Server With Tracing Enabled"
- Section 2.2.2, "Troubleshooting an Address Already In Use Error"
- Section 2.2.3, "Overwriting OracleAS WEB.XML Causes Connection Failure"

2.2.1 Running the Mobile Server With Tracing Enabled

If you experience any difficulty with the Mobile Server running with the application server or the standalone Mobile Server, you can enable tracing in the Mobile Server.

To enable tracing in the Mobile Server, set up your environment as described in Section 3.1, "Enable Tracing on the Mobile Server". To enable tracing on your Mobile Client, follow the instructions in Section 3.2, "Enable Tracing on Mobile Clients".

2.2.2 Troubleshooting an Address Already In Use Error

When you start the Mobile Server, it fails with the following error:

```
ERROR J2EE RMI0002 Error starting ORMI server
```

```
@ 1 port 23,791: Address already in use: JVM_Bind
ERROR java.net.BindException: Address already in
use: JVM_Bind
```

You need to modify the RMI and JMS ports in the `rmi.xml` and `jms.xml` file, which are located in the *<ORACLE_HOME>*`\mobile_oc4j\j2ee\mobileserver\config` directory. The port numbers may not already be in use by another process.

Normally, this error is caused by installing either of the following:

- Two standalone Mobile Servers on a single machine
- A standalone Mobile Server and a Web-to-Go client for OC4J on the same machine

2.2.3 Overwriting OracleAS WEB.XML Causes Connection Failure

When you install OracleAS and Oracle Database Lite; normally, you install the OracleAS first and then Oracle Database Lite. The Oracle Database Lite adds an entry to the `web.xml` file to define Oracle Database Lite as a Web service. This includes a definition and location of the webtogo.ora file. So, if you have reinstalled OracleAS or corrupted the `web.xml` file in any way, Oracle Database Lite and OracleAS do not interact as they should.

The best method is to reinstall Oracle Database Lite on top of a fresh OracleAS installation. Otherwise, if you have a backup copy of the web.xml file, you may only need to overwrite the new one with the backed up copy. Verify that everything is the same except for the addition of a `web-app` definition for `webtogo-web`.

2.3 Troubleshooting the Mobile Server Repository

The following sections describe how to recover the Mobile Server Repository:

- Section 2.3.1, "Troubleshooting the Mobile Server Repository with the Mobile Server Repository and Diagnostic Tool (MSRDT)"
- Section 2.3.2, "Modifying IP Address of Machine Where Mobile Server Repository Exists"

2.3.1 Troubleshooting the Mobile Server Repository with the Mobile Server Repository and Diagnostic Tool (MSRDT)

Customers may modify the Mobile Server repository in the back-end database and, without realizing it, violate some of the rules. We have provided a tool that helps you analyze, validate, and debug the repository.

The Mobile Server Repository Diagnosis tool evaluates the repository and prints out only what is found to be modified or missing. The output generated is best viewed if you set your column width to 80 or if you pipe it into a file, to view it with Word or WordPad.

- Section 2.3.1.1, "Inspecting Files in the Mobile Server Repository"
- Section 2.3.1.2, "Use the Mobile Server Repository and Diagnostic Tool to Validate Your Environment and the Repository"
- Section 2.3.1.3, "Execute the Repository Diagnostics Tool"

2.3.1.1 Inspecting Files in the Mobile Server Repository

You can use the Mobile Server shell utility (`wsh`) to inspect and modify the Mobile Server Repository interactively. Start the Command Prompt and enter the following.

```
wsh -L system/x@olite-db
```

OR

```
wsh -o <adminuser>/<adminuser's password>@o8db
```

For example, you could enter the following sample codes at the command prompt.

```
wsh -o administrator/admin@webtogo.world
wsh -L system/x@webtogo
```

This displays the Mobile Server Repository prompt.

The following table lists commands that are available for inspecting and altering the Mobile Server Repository.

Table 2–1 describes available commands for inspecting and altering the Mobile Server Repository.

Table 2–1 Commands to Inspect and Alter the Mobile Server Repository

Command	Definition
`dir`	Displays a list of files in a directory.
`copy`	Copies one or more files to another location.
`cp`	Copies one or more files to another location.
`edit`	Launches Notepad for editing a file.
`del`	Deletes one or more files.
`rm`	Deletes one or more files.
`cd`	Displays the name or changes the current directory.
`md`	Creates a directory.
`rd`	Removes (deletes) a directory. Use the option `-s` to remove a directory including all subdirectories.
`type`	Displays the contents of a text file or files.
`exit`	Quits the command shell.
`quit`	Quits the command shell.
`help`	Provides help information for shell commands.
`sync`	Synchronizes the file system with the database.

2.3.1.2 Use the Mobile Server Repository and Diagnostic Tool to Validate Your Environment and the Repository

You can use the Mobile Server Repository and Diagnostic Tool (MSRDT) to validate your environment and what is in the back-end Mobile Server repository. The following sections describe the validation that occurs:

- Section 2.3.1.2.1, "Validate the Environment for the Mobile Server"
- Section 2.3.1.2.2, "Validate Integrity of Mobile Server Tables and Data"
- Section 2.3.1.2.3, "Validate the Structure and Contents of the Repository"

2.3.1.2.1 Validate the Environment for the Mobile Server Displays the system configuration environment for the host where the Mobile Server resides, as follows:

- Web-to-Go version
- Consolidator version
- Back-end Oracle database version
- Definition of the Java library path
- Definition of the Java `CLASSPATH`
- Operating system architecture, type and version
- Java VM vendor, version, name, home, and info (mode)
- File encoding used
- Path separator and file separator used
- Country, time zone, and user language
- Lists the Mobile Server administrators
- Lists the contents/current configuration for the webtogo.ora file

2.3.1.2.2 Validate Integrity of Mobile Server Tables and Data When Mobile Server and the repository are installed, certain tables, constraints, objects, and so on cannot be modified. This part of the diagnostic tool checks that these requirements are still the same as when installed, which includes the following:

- Required tables exist
- All columns within required tables exist
- Required table attributes exist
- Required constraints exist or have not been modified
- Required sequences exist
- Extra tables have not been added to Mobile Server schemas
- Web-to-Go system and its application publications exist
- Application files integrity check: If you have published an application to the server, then check if the folder exists and is not empty. Also, if sharing a repository among multiple Mobile Servers, ensures that the application is published on this Mobile Server, where this tool is executed.

2.3.1.2.3 Validate the Structure and Contents of the Repository If you are experiencing trouble with your repository, execute this tool to determine if any of the following have occurred:

- The mapping between primary keys for the map table and corresponding base table must be consistent.
- Invalid indexes
- Consistency of the In-Queue schema with the view schema.
- The records in the in-queue master table have corresponding records in the in-queue detail table.
- Any DML locked tables or entries in the `C$ALL_LOGGED_TABLES`.
- Invalid triggers.

- Validate that all of the users subscribed to a publication also have all of the corresponding sequences assigned to the same publication.
- Invalid sequence values.
- Validate that every column in C$ALL_SEQUENCES has its own table, which is named C$WD_<column>.
- Validate that every window sequence has the same id as a record in the C$ALL_SEQUENCES.
- Look for any orphaned objects in the repository.
- Verify that the Mobile Server repository owner is granted with sufficient privileges.
- Validate the MGP properties: If MGP_SUSPEND(apply suspend) is false, then MGP_RUN must be true.
- Reports on the most recent jobs and their status.

2.3.1.3 Execute the Repository Diagnostics Tool

The following is the usage and syntax for the Mobile Server repository diagnostics tool—msrdt:

```
msrdt <options>
```

Where <options> are the following:

Table 2–2 List of Options for Repository Diagnostic Tool

Tasks	Options	Description
Validate Repository	-v <username>/<pwd>@ <hostname>:<port>:<SID>	Validates the repository and provides error reporting. Performs some error recovery.

2.3.2 Modifying IP Address of Machine Where Mobile Server Repository Exists

During the installation, the machine name or IP address is provided by the user where the repository is created. If the IP address of the machine changes, then perform one of the two options:

- If the user provided the machine name; then even after the IP change, the machine name will still work.
- If user provided the IP address—instead of machine name—then after changing the IP address of the repository machine, the user must change the ADMIN_JDBC_URL and THIN_JDBC_URL parameters in the webtogo.ora file on the Mobile Server.

2.4 Troubleshooting the Oracle Lite Databases

The following sections detail how to either work with the client database or modify an existing schema on the back-end Oracle database:

- Section 2.4.1, "Accessing the Client Database Offline"
- Section 2.4.2, "Determining Source of Checksum Error Against Database"

2.4.1 Accessing the Client Database Offline

When you go offline, after being online, and you need to access your client database, use your Mobile user password, not the password `manager`. It is the password that you enter when you logged in, before going offline.

If you do not use your Mobile user password when you try to access your client database offline, you will receive the following error message.

```
[POL-5150] access violation
```

For example, if you log in as `JOHN/JOHN` on a Windows 32 machine with contains a Mobile Client for Web-to-Go, and `JOHN` goes offline, the user database created for JOHN on the client machine requires `JOHN`'s password for access.

If you use mSQL to access the user database, start the Command Prompt and enter the following statement.

```
msql system/john@jdbc:polite:john_todo
```

In this example, `JOHN`'s password is `john`, and the DSN name that was created automatically on the client for the user name `JOHN` is `john_todo`.

Syntax

The following syntax enables you to access the user database.

```
msql system/<mobile user's password>@jdbc:polite:<dsn created for the user>
```

To find the DSN name, use the ODBC Admin tool on your client machine.

2.4.2 Determining Source of Checksum Error Against Database

You can perform diagnostics if you experience database corruption due to file system write errors, I/O errors, or a media device problem. If you receive a POL-3207 error, you may wish to execute the validatedb tool to see if it is a checksum error. Then, setting `OLITE_WRITE_VERIFY` to `TRUE` generates error reporting if a checksum error occurs on the device for the Mobile client.

For more information, see Section G.2.15 "OLITE_WRITE_VERIFY" in the *Oracle Database Lite Administration and Deployment Guide*.

2.5 Troubleshooting JVM Errors

This section focuses on how to debug the following Java errors:

- Section 2.5.1, "Troubleshooting An Out of Memory Error"
- Section 2.5.2, "Troubleshooting an IllegalArgumentException"

2.5.1 Troubleshooting An Out of Memory Error

The Mobile Server executes within the OC4J container, which in turn runs on a Java virtual machine (JVM). When you are experiencing the `OutOfMemory` error, then you should have an understanding of JVM memory architecture and OC4J when tuning Mobile Server performance.

The following may cause an `OutOfMemory` error:

- A memory leak in the Mobile Server
- Not enough physical memory to handle your application

- In-appropriate allocation of the three memory areas that used by the JVM. See Section 2.5.1.1, "JVM Memory Settings" for a full description.
- Memory being held by the Mobile Server. See Section 2.5.1.3, "Why is Memory Not Released?" for more information.
- Understanding how threads are consuming your memory. See Section 2.5.1.4, "Thread Memory Consumption and Concurrency" for full details.

2.5.1.1 JVM Memory Settings

JVMs may have different implementations of memory management and garbage collection schemes. But at a higher level, they all arrange the memory in the following three areas:

- Section 2.5.1.1.1, "Java Heap"—The Java heap is where the Java objects live. It is normally the largest of the three.
- Section 2.5.1.1.2, "Permanent Generation"—This is where the classes are loaded.
- Section 2.5.1.1.3, "Native Space"—The memory used by native code, which includes JVM native code and application JNI calls.
- Section 2.5.1.1.4, "Setting Java Options for Java Memory"—Where you set these options depends on your environment.

This section describes how to modify the allocation of memory to the JVM memory areas.

> **Note:** Memory should be allocated properly for the three areas. Otherwise, different kinds of `OutOfMemory` error may surface.

2.5.1.1.1 Java Heap The Java heap is where Java objects live. It consists of both the young and tenured generations. The amount of Java heap memory that the JVM starts with is designated by the initial heap size option (`-Xms`) and the maximum heap size option (`-Xmx`).

If you see from the stack trace that a Java method throws an `OutOfMemory` error, then you have exhausted your Java heap space.

For example:

```
java.lang.OutOfMemoryError: Java heap space
```

The default settings for the Java heap for a SunUNIX JVM is as follows: `Xmx:64M Xms:4M`. However, the default for the Mobile Server—if you start up the Mobile Server with the `runmobileserver.bat` executable—is set to `Xmx:256M Xms:512M`.

The size of the space reserved can be specified with the `-Xmx` option. The `-Xms` specifies the space that is immediately committed to the virtual machine. We recommend to allocate ¼ to ½ of the available physical memory to Java Heap. If you set the maximum Java Heap size to be large—such as, 512M—and you still receive this error, then there may be a leak in the Java code.

The following shows the syntax for how to increase the amount of Java heap memory available to the JVM through the `-Xms` and `-Xmx` Java switches:

```
java -Xms<memory_size>m -Xmx<memory_size>m -jar oc4j.jar
```

The amounts specified should be based on the available resources. At the minimum, you should set both values to at least 256 MB. The following allocates 768M for both

the initial and maximum heap sizes. Of course, the amount of memory you allocate depends on what you have available.

```
java -Xms768m -Xmx768m -jar oc4j.jar
```

> **Note:** Set the Java heap memory size before starting the Mobile Server with the `runmobileserver` executable.

2.5.1.1.2 Permanent Generation The permanent generation holds data needed by the JVM that describes objects which do not have an equivalence at the Java language level. For example, permanent generation is where the classes are loaded. It holds objects that describe classes and methods.

If a classloader method or a String intern method throws an `OutOfMemory` error in the stack trace, then you have run out of permanent generation space.

For example:

```
java.lang.OutOfMemoryError: PermGen space at
java.lang.ClassLoader.defineClass1(Native Method) at
java.lang.ClassLoader.defineClass(ClassLoader.java:620)
```

The default for the permanent generation for a Sun UNIX JVM is 64MB. To set a new initial size for the Sun JVM, use the `-XX:PermSize` option when starting the virtual machine. To set the maximum permanent generation size use the `-XX:MaxPermSize` option.

2.5.1.1.3 Native Space The native space is the memory used by native code, which includes JVM native code and application JNI calls. If a native method throws an `OutOfMemory` error or the JVM crashes with such an error, then you run out of native space.

For example:

```
java.lang.OutOfMemoryError: requested 14892 bytes. Out of swap space?
java.lang.OutOfMemoryError: unable to create new native thread
```

The native space is the (Available physical memory) – (Java heap + Permanent generation). There is no way to set the native space, except to decrease the Java heap or permanent space. If you allocate too much memory for the Java heap, then the native code is left with not enough memory and may run out. If you have to increase the native memory, then decrease the `-Xmx` parameter to a reasonable value to leave enough memory for the native space. If you still get this error, the native code may have a memory leak.

2.5.1.1.4 Setting Java Options for Java Memory Set the Java options when you start the OC4J container. The following example sets the initial Java heap to 256M, the maximum Java heap to 512M, and the permanent generation memory to 64M:

```
java –Xms=256m –Xmx=512m -XX:MaxPermSize=64m oc4j.jar
```

You set the Java options depending on the environment, as follows:

- The `JAVA_OPTIONS` parameter in the `WEBTOGO.ORA` file is used only for client-side settings. This parameter is ignored on the server-side.
- For the Mobile Server, all modifications for the Java options must be specified on the command-line or in the `runmobileserver.bat` file.

- When modifying the Java memory options for OracleAS, modify the JVM settings in the `opmn.xml` file, which is located in the *<IAS_HOME>*`/opmn/conf` directory.

  In the `opmn.xml` file, search for `process-type id="mobileserver"` and then modify the `java-options` parameter under `category id="start-parameters"`, as shown below:

```
<process-type id="mobileserver" module-id="OC4J">
 <module-data>
 <category id="start-parameters">
 <data id="java-options" value="-server -Xrs
   -Djava.security.policy=/j2ee/mobileserver/config/java2.policy
   -Djava.awt.headless=true"/>
 <data id="oc4j-options" value="-properties"/>
```

By default, the MGP executes as a job in the Job Scheduler in the Mobile Server. Thus, the MGP and other Mobile Server components, such as the Sync Server, share the same memory space. This provides efficiency and manageability; however, if the MGP has a memory leak, then the Mobile Server is affected. In this case, perform the following:

1. Disable the MGP job.
2. Restart the Mobile Server.
3. Restart the MGP in a separate JVM with either the `mgp.bat` or, if using UNIX, the `mgp` shell script. This JVM is restarted periodically and may hide the memory leak issue.

> **Note:** Set the Java memory options for the MGP in the `mgp.bat` file.

With a larger Java heap size, the garbage collector collects less often and consumes less CPU time. Therefore, a larger heap size is desired for better performance. For the Mobile Server, most of the code is Java code; for the Oracle Lite JDBC connection and Java mSync client, most of the code is in native code. So, setting the Java heap size larger, helps the efficiency and performance of the Mobile Server; however, if it is set too high, the JDBC connection and mSync client may have memory issues.

2.5.1.2 Modifying Java Options for Java Memory When Using Oracle AS

If you are using Oracle AS, then set the JVM settings for your Java memory in the `opmn.xml` file, which is located in the *<IAS_HOME>*`/opmn/conf/` directory. Inside the opmn.xml file, perform the following:

1. Modify `process-type id="mobileserver"`.
2. Change `java-options` under `category id="start-parameters"`.

The following is an example of this modification:

```
<process-type id="mobileserver" module-id="OC4J">
   <module-data>
      <category id="start-parameters">
          <data id="java-options" value="-server -Xrs
              -Djava.security.policy=/j2ee/mobileserver/config/java2.policy
              -Djava.awt.headless=true"/>
           <data id="oc4j-options" value="-properties"/>
```

2.5.1.3 Why is Memory Not Released?

You may expect Mobile Server to release the free memory back to operating system after it has finished its work. However, the Mobile Server holds a large amount of

memory even when it is idle. This may not be an indication of memory leak; instead, it may be for one or more of the following reasons:

- If you set the `-Xms` option to a large number—such as, 1024M—then you should expect the Mobile Server process to use at least 1024M until the process is killed.
- For performance reasons, the Mobile Server caches metadata in Java heap memory.
- The garbage collector may not collect the objects right way when they are no longer referenced. In addition, the garbage collector keeps a large amount of free memory in the Java heap for future allocations, instead of returning them to the operating system. You can use Java options to adjust the free memory size; instead, view the Mobile Server total runtime Java heap size and free heap size in the Mobile Manager at Mobile Manager->Data Synchronization->Host.

2.5.1.4 Thread Memory Consumption and Concurrency

The Java heap and permanent generation together are called managed heap, since the garbage collector manages them. The native space can be divided into native heap and thread stack space. Each thread consumes memory, as follows:

- Each thread created consumes about 1MB stack space, although it is JVM dependent. Take this memory into consideration if you execute multiple threads. For example, on a 32-bit x86 system, the (managed heap + native heap + thread stack size * number of threads) could not exceed 2 GB. On any system, ensure that the total JVM memory is less than the available physical memory size.
- Each thread allocates additional Java and native heap memory as it executes.
- There is an overhead associated with multi-threading. Therefore, be careful when executing too many concurrent threads. If concurrency is set to larger than 20, then you are more likely decreasing the Mobile Server throughput—instead of increasing it.

You can control the concurrency of Mobile Server; that is, the OC4J layer and the synchronization layer, as follows:

- Section 2.5.1.4.1, "Configure the Number of HTTP Sessions"
- Section 2.5.1.4.2, "Configure the Synchronization Session Thread Concurrency and MGP Thread Maximum"

2.5.1.4.1 Configure the Number of HTTP Sessions Set the OC4J `max-http-connections` parameter in the `server.xml` file, which is located at `<ORACLE_HOME>/j2ee/mobileaserver/config/`. This OC4J parameter affects all HTTP sessions, which includes Mobile Manager Web requests.

For example, the following sets the `max-http-connections` parameter to 10:

```
<application-server … … >
   <max-http-connections max-connections-queue-timeout="180"
      socket-backlog="50" value="10">
   </max-http-connections>
…
</application-server>
```

The following table describes the attributes of the `max-http-connections` parameter:

Table 2–3

Attribute	Description
`max-connections-queue-timeout`	Specifies the number of seconds to wait for an available connection. The connection may be waiting if the maximum connections has been reached before returning either "server busy" or "redirect" messages. Default value: 10 seconds
`socket-backlog`	Specifies the number of connections that can be queued up before denying connections at the socket level. Default: 30.
`value`	Specifies the maximum number of connections. Default: 100000.

2.5.1.4.2 Configure the Synchronization Session Thread Concurrency and MGP Thread Maximum To control the synchronization session thread concurrency, set the Consolidator parameters `MAX_CONCURRENT` and `MAX_CONCURRENT_TIMEOUT`. To control the number of threads spawned within the MGP process, set the Consolidator parameter `MAX_THREADS`. See Section A.6, "CONSOLIDATOR" in the *Oracle Database Lite Administration and Deployment Guide* for more information on these parameters.

2.5.2 Troubleshooting an IllegalArgumentException

If you receive the `java.lang.IllegalArgumentException: Signal already used by VM:SIGINT` exception and you are using Branch Office, then you are using a JRE version previous to version 1.4.2. Update to JRE version 1.4.2 or later.

2.6 Troubleshooting Security

The following section describes how to troubleshoot security issues:

- Section 2.6.1, "SSL Certificate Rejection for Client Authentication"

2.6.1 SSL Certificate Rejection for Client Authentication

If you are using a reverse proxy and have configured SSL between the client and the reverse proxy, you may receive the following error:

```
A certificate is required to complete client authentication.
```

For all clients, except Web-to-Go clients, you can only use SSL authentication with a signed certificate. If you use a self-signed certificate, you must turn off SSL authentication by adding the following to the `NETWORK` section in the client `polite.ini` file:

```
DISABLE_SSL_CHECK=YES
```

This parameter tells the reverse proxy firewall to use SSL encryption for the communication from the client, but not to perform SSL authentication.

3

Tracing and Logging

You can enable tracing for the Mobile Server, the Mobile Client, or the Oracle Lite database on the client. In addition, you can view the log files from the underlying application server. The methods for enabling tracing for each component is described in the following sections:

- Section 3.1, "Enable Tracing on the Mobile Server"
- Section 3.2, "Enable Tracing on Mobile Clients"
- Section 3.3, "Enabling Tracing in the Client-Side Oracle Lite Database"
- Section 3.4, "Viewing the Log Files From the Application Server"

3.1 Enable Tracing on the Mobile Server

For the Mobile Server, there are two main sections for tracing: the general tracing for Mobile Server components and specific tracing for data synchronization components. How to enable tracing for each part of the Mobile Server is described in the following sections:

- Section 3.1.1, "General Tracing for the Mobile Server"
- Section 3.1.2, "Data Synchronization Tracing"

3.1.1 General Tracing for the Mobile Server

To set general tracing for the Mobile Server, perform the following steps.

1. From the Mobile Server page, select **Administration**.
2. Select **Trace Setting**. This brings up the Trace Settings page, as shown in Figure 3–1, where you can choose to generate trace output, specify the trace output destination to the local console, file, or remote console (viewed by `wsh`). The Trace Settings page provides system filters to generate trace output to the required system level.
3. Configure the type of tracing you want and click **Apply**.

Figure 3–1 General Trace Settings for Mobile Server

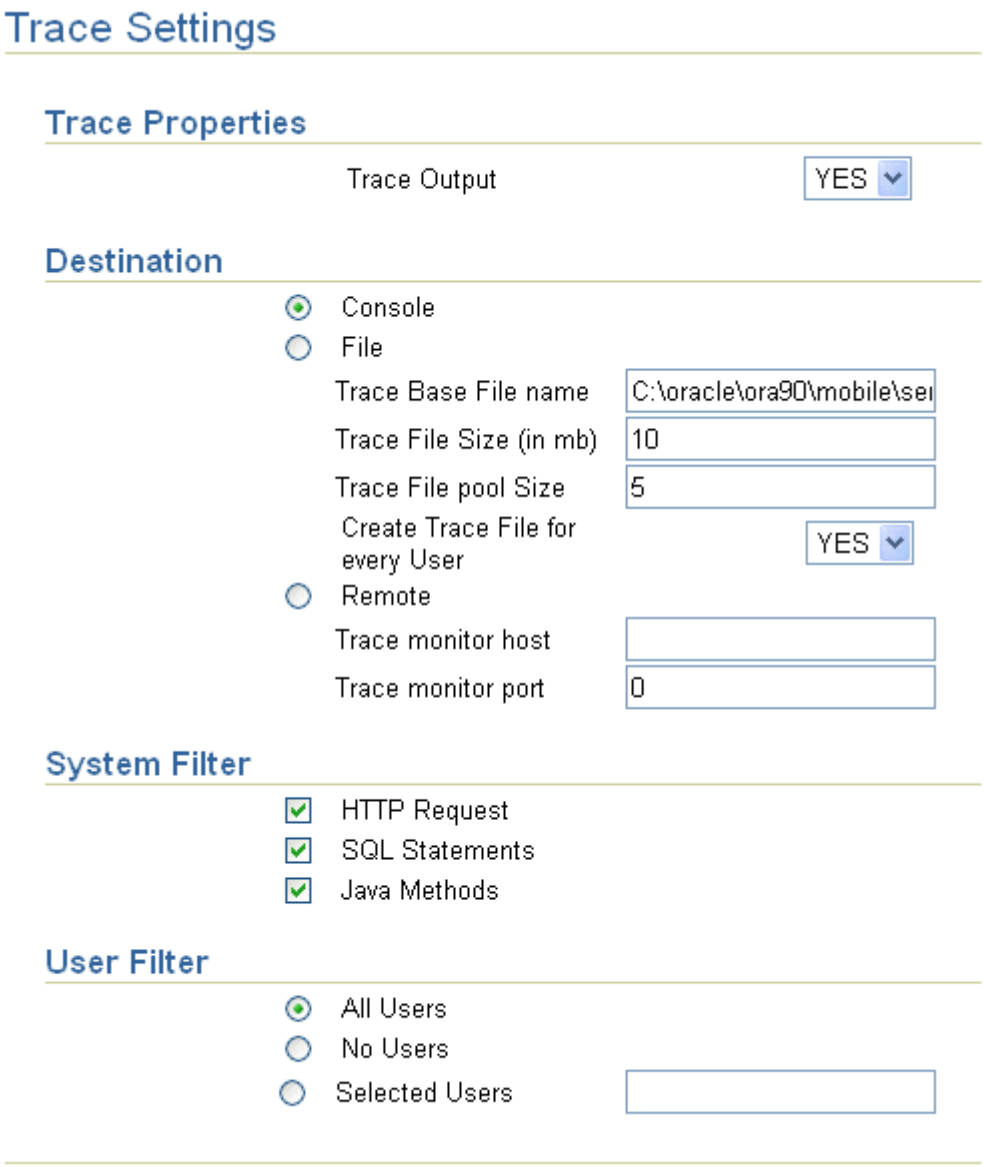


Table 3–1 Trace Settings Page Description

Field	Description
Trace Output	To generate trace output, select **Yes**.
Console	You can print the messages to a console. You can ONLY choose the console if you are executing Mobile Server in standalone mode. If you are in an OracleAS environment, select File or Remote.

Table 3–1 (Cont.) Trace Settings Page Description

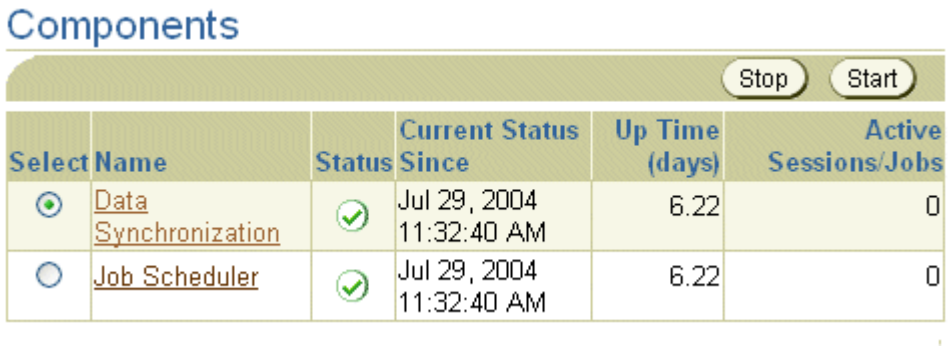
Field	Description
File	You can direct all messages to a local file. If you selected a file for trace output, then enter the name (including path), the maximum size of the file in MB, and the number of files allowed (pool size). For example, if you set the pool size to 10, then when a trace file hits the maximum size in MB, then a new file is opened and the trace output is written to the new file. This continues until all 10 files of the maximum size exist. At this point, the first file is deleted and a new file is started to contain the trace output. This enables you to manage the amount of disk space that the trace files can use. To create a trace file for every user, select **Yes** for the Create Trace File for Every User box.
System Filter	▪ HTTP Request—To generate HTTP output and Web-to-Go trace information as trace output, select this option. This includes general system information. ▪ SQL Statements—To generate SQL queries as trace output, select this option. ▪ Java Methods—To generate all `system.out` output from the Mobile Server and Data Synchronization Java methods, select this option. **Note**: The Mobile Server automatically filters exceptions and errors as trace output at the Mandatory level.

3.1.2 Data Synchronization Tracing

The administrator can turn on tracing for components involved in the synchronization phase, including MGP functions.

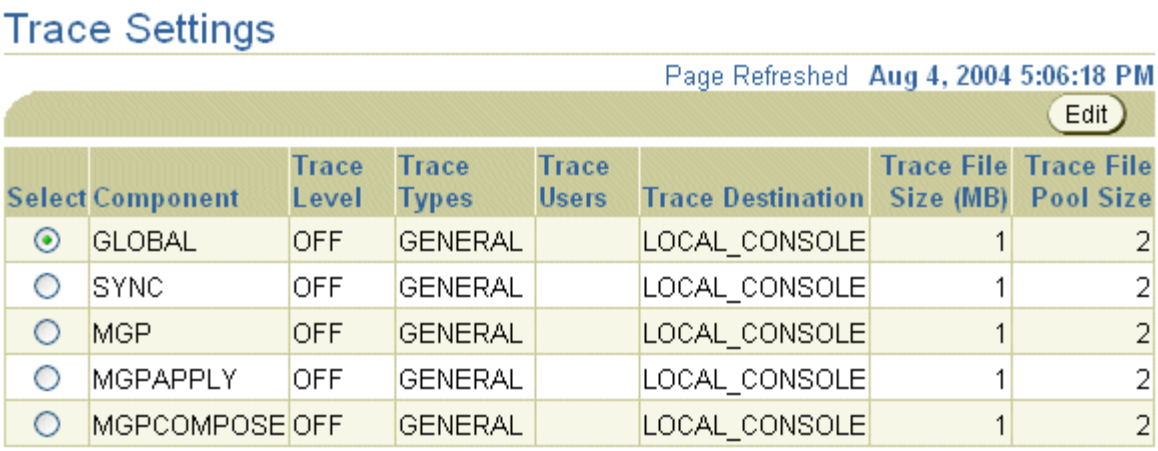
1. From either the home page or the Administration page for the Mobile Server, select **Data Synchronization** in the Components section, as shown in Figure 3–2.

Figure 3–2 Mobile Server Job Scheduler and Data Synchronization Components



2. Select **Administration** off of the Data Synchronization home page.
3. Select **Trace Settings**, which displays all five components for which you can enable tracing, as shown in Figure 3–3. For a description of each component, see Section 3.1.2.1, "Description of the Five Data Synchronization Components".

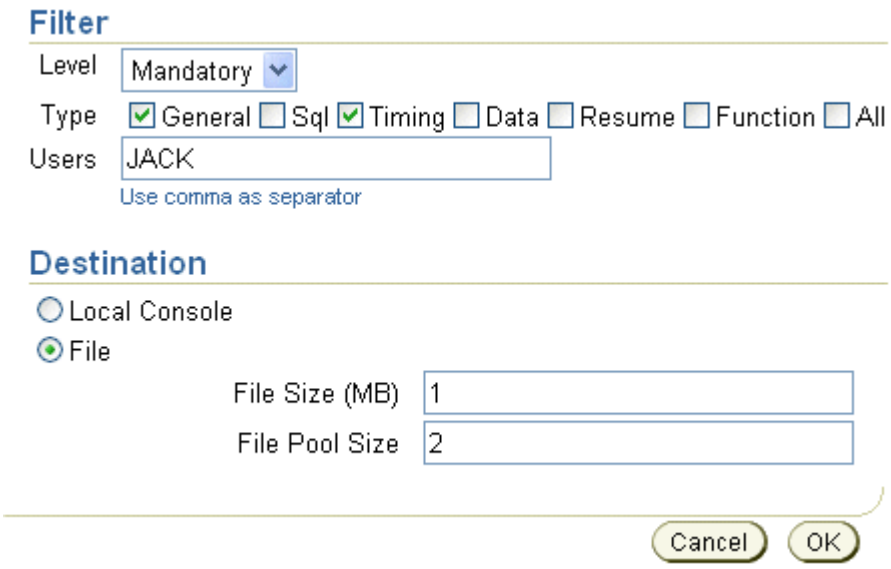
Figure 3–3 The Trace Components for the Data Synchronization



Select	Component	Trace Level	Trace Types	Trace Users	Trace Destination	Trace File Size (MB)	Trace File Pool Size
◉	GLOBAL	OFF	GENERAL		LOCAL_CONSOLE	1	2
○	SYNC	OFF	GENERAL		LOCAL_CONSOLE	1	2
○	MGP	OFF	GENERAL		LOCAL_CONSOLE	1	2
○	MGPAPPLY	OFF	GENERAL		LOCAL_CONSOLE	1	2
○	MGPCOMPOSE	OFF	GENERAL		LOCAL_CONSOLE	1	2

4. Select the component for which you want to enable tracing, which brings up the trace configuration screen, as shown in Figure 3–4.

Figure 3–4 Data Synchronization Component Trace Configuration



- In the Filter section, select the required **Level** and **Type**. To specify a trace filter for users, enter comma separated user names in the **Users** field.

Table 3–2 Data Synchronization Component Trace Level and Type

Filter	Description
trace level, where each level includes the previous levels as well.	OFF: no tracing enabled. MANDATORY: Mandatory messages only, such as program exceptions. WARNING: Warning messages. NORMAL: Normal messages of which the user must be informed. INFO: Informational messages, such as synchronization timing, MGP apply, MGP compose, and MGP status. CONFIG: Configuration messages, such as JDBC driver version. FINEST: Developer level of tracing. ALL: Logs messages for all trace levels.
trace type	SQL: SQL-related messages only, such as SQL statements. TIMING: Timing data only. **Note**: This option is trace level sensitive. For MGP Cycle time and Synchronization time, use the Trace Level INFO option with the TIMING option on the MGP and SYNC components respectively. DATA: Data only. RESUME: Logs messages with Reliable Transport. FUNCTION: Displays the program flow by logging methods such as Entry, Exit or Invoke. For Long methods, this option logs the method entry or exit; which is a simple invoke log. GENERAL: Logs messages that do not belong to any of the above listed trace types. **Note**: This type is trace level sensitive. ALL: This option generates logs of all trace types.

Note: You can set these parameters within the webtogo.ora file in the CONSOLIDATOR section. See Section A.6, "[CONSOLIDATOR]" in the *Oracle Database Lite Administration and Deployment Guide* for more details on these parameters.

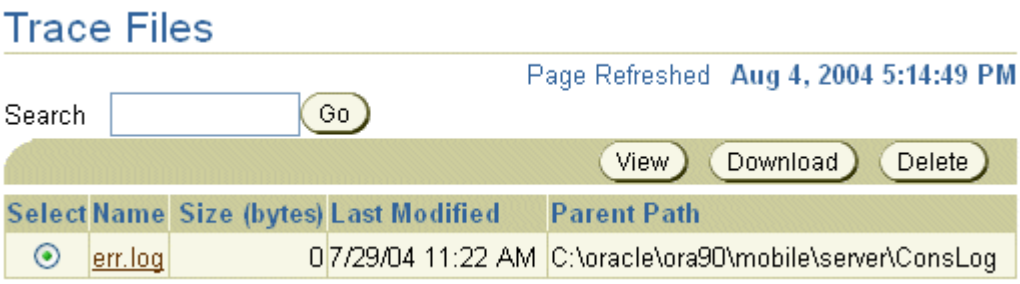
- In the Destination section, select **Local Console** to receive the trace file to the same console as the General tracing is using. If the console is not open, then these messages are sent to the same place that the General tracing is directed. See what the Destination is configured to in Figure 3–1 to determine where these messages are directed.

  To send trace information to a file, select the **File** option. The file name is generated based upon the session id. You can configure the file size in MB and the files allowed (pool number). For example, if you set the pool size to 10, then when a trace file hits the maximum size in MB, then a new file is opened and the trace output is written to the new file. This continues until all 10 files of the maximum size exist. At this point, the first file is deleted and a new file is started to contain the trace output. This enables you to manage the amount of disk space that the trace files can use.

5. To implement the modified values, click **OK**. To retain existing values, click **Cancel**.

To view trace files, navigate to the Data Synchronization page. Select **Administration**. Select **Trace Files** and the Trace Files screen appears, as shown in Figure 3–5.

Figure 3–5 Viewing Data Synchronization Trace Files



- To view a trace file, select the trace file name or click the Select button next to the trace file name and click **View**.

> **Note:** When you view the trace file online, it truncates the file to 10,000 lines. To view the whole trace file, download the file and view it using any text editor.

- To download or delete a trace file, click the Select button next to the trace file name and click either **Download** or **Delete**.
- If there are too many files to view on a page, you can search by entering the name of the trace file in the Search field and clicking **Go**.

3.1.2.1 Description of the Five Data Synchronization Components

There are five components that you can turn on to describe what is happening in the synchronization process, as described in the following sections:

- Section 3.1.2.1.1, "MGP"
- Section 3.1.2.1.2, "MGPAPPLY"
- Section 3.1.2.1.3, "MGPCOMPOSE"
- Section 3.1.2.1.4, "SYNC"
- Section 3.1.2.1.5, "GLOBAL"

3.1.2.1.1 MGP You can trace the MGP process. However, if an `MGP Cycle ID` is not yet available, then tracing is enabled by the configuration of the `GLOBAL` component. If the trace destination is to be written to a file, then all of the generated logs are recorded in a log file named `MGP_<cycle_id>.log`.

3.1.2.1.2 MGPAPPLY This refers to the `APPLY` phase in the MGP process. However, between the beginning of the `APPLY` phase till the availability of the MGP Client ID, tracing is enabled by the configuration of the component MGP. If tracing is sent to a file, then all messages are written to a file named `MGPAPPLY_<client_id>_<cycle_id>.log`.

3.1.2.1.3 MGPCOMPOSE This refers to the `COMPOSE` phase in the MGP process. Similar to the `MGPAPPLY` phase where the Client ID is not yet available, tracing is enabled by the configuration of the component MGP. If tracing is sent to a file, then all messages are written to a file named `MGPCOMPOSE_<client_id>_<cycle_id>.log`.

3.1.2.1.4 SYNC This refers to the server-side synchronization process. When a `Sync session ID` is not yet available, tracing is enabled by the configuration of the

GLOBAL component. If the trace destination is set to file, then the messages are written to a file named SYNC_<cycle_id>.log. When the Client ID becomes available, the file is renamed to SYNC_<client_id>_<cycle_id>.log.

3.1.2.1.5 GLOBAL This component logs tracing messages that are not specific to any of the above listed components. This component also includes logs that are generated during the execution of the ConsolidatorManager APIs. If the trace destination is set to file, then the messages are written to a file named GLOBAL_<file_number>.log.

3.2 Enable Tracing on Mobile Clients

You can also enable tracing on your Mobile client through one of the following methods:

- Section 3.2.1, "Turn on Tracing using the Mobile Client WEBTOGO.ORA File"
- Section 3.2.2, "Turn on Tracing using the -d0 Option for Web-to-Go Clients With the WEBTOGO Executable"
- Section 3.2.3, "View Device Logs"

3.2.1 Turn on Tracing using the Mobile Client WEBTOGO.ORA File

You can enable tracing through the DEBUG section in the webtogo.ora file on your Mobile Client. This is only valid for Mobile Client for the Web (Web-to-Go), Branch Office, andBC4J clients. Restart your Mobile client after modifying the webtogo.ora file to enable tracing.

See Section A.3, "[DEBUG]" in the *Oracle Database Lite Administration and Deployment Guide* for a full description of the trace parameters. Each trace parameter matches the fields displayed on the General trace settings screen for the Mobile Server, as shown in Figure 3–1.

3.2.2 Turn on Tracing using the -d0 Option for Web-to-Go Clients With the WEBTOGO Executable

If you want to enable tracing quickly to a console window on the Web-to-Go Mobile client, execute the Mobile client webtogo command with the -d0 option. With the -d0 option, tracing is enabled and printed to a console window. The level of tracing shown is indicated by the TRACE_LEVEL parameter in the DEBUG section of the webtogo.ora file. All other DEBUG parameters are ignored in this situation. In order to start the Mobile client with the -d0 option, you must first stop your existing client.

You can only use this type of tracing for Mobile Client for the Web, Branch Office, and BC4J.

For more information on configuring the TRACE_LEVEL parameter in the webtogo.ora file, see Section A.3, "[DEBUG]" in the *Oracle Database Lite Administration and Deployment Guide*.

3.2.3 View Device Logs

Each Mobile device maintains a log of the activity that it generates. See Section 8.4.7, "Viewing Device Logs" in the *Oracle Database Lite Administration and Deployment Guide* for more information.

3.3 Enabling Tracing in the Client-Side Oracle Lite Database

The Oracle Lite database is used in conjunction with other products such as Oracle forms, SQLJ, Web Servers, and OC4J. When an unexpected error is reported by the software system, users need to identify the location and cause of the error. Errors can be caused due to problems in the application code, Oracle tools—such as forms, SQLJ, OC4J—or in the Oracle Lite database. Errors also occur in simple environments where a user application talks directly to the Oracle Lite database through JDBC or ODBC drivers. It may not be obvious which component is at fault—whether it is the user application, JDBC or ODBC drivers, or the core database runtime system.

If the optimizer spends too much time evaluating alternative plans or collecting index statistics, a query may take a long time for compilation. If the execution plan selected by the optimizer is not optimal, the query may also take a long time during execution. Based on these criteria, the tracing facility provides the compilation time and the execution plan.

The following sections describe how to set and use tracing.

- Section 3.3.1, "Enabling Trace Output"
- Section 3.3.2, "Description of Trace Information"

3.3.1 Enabling Trace Output

To enable Trace output, include the following line in the `POLITE.INI` configuration file:

```
OLITE_SQL_TRACE= yes
```

> **Note:** Any value other than "yes" disables the tracing feature. The parameter value is checked once during database startup. Hence, users must set this value before connecting to the database.

When you enable tracing, the trace information is dumped to a file named `oldb_trc.txt` in the current working directory of the database process. If the file already exists, then the trace output is appended to the end. If it does not exist, then a new file is automatically created. For a database service on Windows or the Oracle Lite database daemon for a Linux platform, the current working directory is specified by the `wdir` parameter during startup of the database service or daemon.

> **Note:** To implement the tracing feature, the database process must contain permissions to create the trace file in the current working directory.

3.3.2 Description of Trace Information

The following trace information is provided:

Table 3–3 Trace Output

Trace Output	Description
Statement Text	Each time a SQL statement is prepared, its text is dumped into the trace file. The SQL statement itself is output without any formatting. If a SQL statement contains a new line character, it is also included in the SQL statement output.

Table 3–3 (Cont.) Trace Output

Trace Output	Description
Compilation Time	After the SQL statement is compiled, the compilation time is printed.
Execution Plan	If there are no errors, the execution plan is printed when available. Only statements that contain a `WHERE` clause generate an execution plan. The printed plan contains the execution order of tables for each sub-select.
Bind Value	If a SQL statement contains markers, then the bind value is printed for every line. Each value for the marker or bind variable is printed on a separate line in the following format. `Marker [<number>]: <Value>` Where, `<number>` is the number of the marker and `<value>` denotes the value of the marker before execution.
Temporary Table Created	Each time a temporary table is created, its name is dumped into the trace file.
Table Access	Each time a table is accessed, the following information is dumped into the trace file: ■ Table Name: The name of the table been accessed is dumped into the trace file. ■ Access Method: The access method used by the database is dumped into the trace file. For a description of how this information is presented, see Section 3.3.2.1, "Table Name Output".
Temporary Table Sorted	Each time a temporary table is sorted, its name and sorting time (in milliseconds) are dumped into the trace file.
First Fetch Time	If the SQL statement is a SELECT statement, the time spent on fetching the first row is dumped into the trace file.
Tid	The thread ID is dumped into the trace file in front of some of the dumped information. The thread is displayed in the following format: `Tid: <thread id>`

3.3.2.1 Table Name Output

The name of the table that is currently being accessed and the method used to access the table are printed in the following formats.

- If the table is accessed sequentially, the format is:

  Table Name: `<table name>`

  Access Method: `Sequential`

  Where `<table name>` is the name of the table being accessed.

- If indices are used, the format is:

  Table Name: `<table name>`

  Access Method: `Term[<number>], Index No: <index number>, IndexName: <index name>`

  `<table name>` is the name of the table being accessed.

`Term[<number>]` is the internal representation of the conjunct search conditions in the WHERE clause.

`<index number>` is the index number. Each index has an unique number in the database.

`<index name>` is the name of the index if any.

3.4 Viewing the Log Files From the Application Server

Since Mobile Server uses OC4J as its application server, you can view the following log files to debug problems.

- Viewing OC4J server level output messages.

  `<OC4J_HOME>\log\server.log`

- Viewing HTTP requests handled by the server.

  `<OC4J_HOME>\log\http-web-access.log`

- Viewing exceptions or errors that are handled by OC4J.

  `<OC4J_HOME>\application-deployments\webtogo\application.log`

- Viewing the file `trace_sys1.log` and other log files that are generated by the Mobile Server in the same directory.

  `<OC4J_HOME>\application-deployments\webtogo\trace_sys1.log`

4

Backup and Recovery

Performing backup and recovery for Oracle Database Lite is the same as what you would normally do for Oracle database applications. The following sections help you understand how to use the Oracle database backup and recovery methods for preserving your Mobile Server and Mobile applications:

- Section 4.1, "How Does Oracle Database Lite Store its Information?"
- Section 4.2, "Backing Up Oracle Database Lite"
- Section 4.3, "Oracle Database Lite Backup Coordination Between Client and Server"
- Section 4.4, "Oracle Database Lite Recovery Issues"

4.1 How Does Oracle Database Lite Store its Information?

Oracle Database Lite uses the Oracle database to store information, as follows:

- The Mobile Server itself is installed and configured as a database application. Thus, the Mobile Server stores its metadata and client state information within a database schema.
- For each Mobile application, the Mobile Server installs triggers and stores transaction data in a schema for that application.

4.2 Backing Up Oracle Database Lite

Since all of the data needed for a backup and recovery strategy exists in the database, you should use the Oracle database backup and recovery strategies discussed in the following books:

- *Oracle Backup Installation Guide*
- *Oracle Database Recovery Manager Quick Start Guide*
- *Oracle Database Backup and Recovery Basics*
- *Oracle Backup Administrator's Guide*
- *Oracle Database Backup and Recovery Advanced User's Guide*

Note: In the past, we recommended that you use export/import to perform a backup. This is not a recommended option anymore. Use the normal online Oracle database backup procedure.

However, the following sections describe what to keep in mind when coming up with a backup and recovery strategy for your Oracle Database Lite environment: Section 4.3, "Oracle Database Lite Backup Coordination Between Client and Server" and Section 4.4, "Oracle Database Lite Recovery Issues".

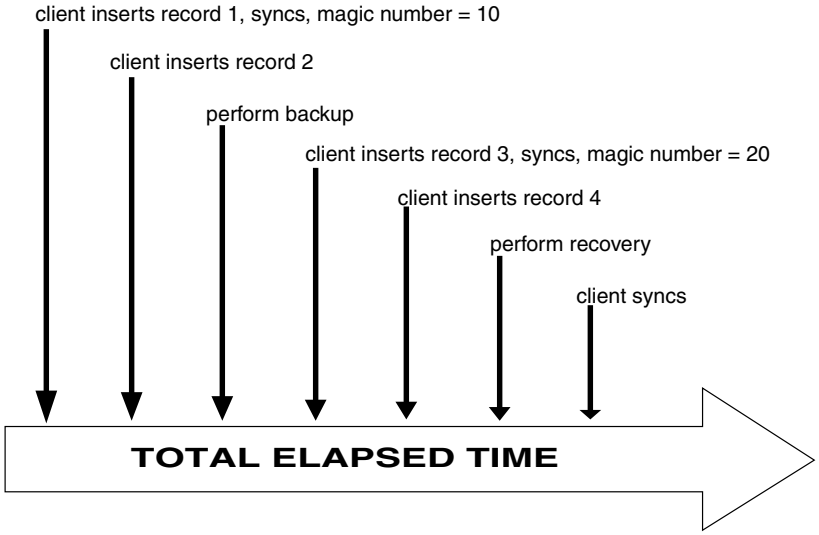
4.3 Oracle Database Lite Backup Coordination Between Client and Server

When a client and a server synchronize with each other, the Mobile Server assigns the same "magic" number to both sides to indicate that the data is in-sync. If this number is different on both sides, the Mobile Server knows that the data is out of sync and is in an error condition. The following example details how this could effect your attempts for a clean recovery.

When you perform a backup, you may lose client data unless you plan accordingly. Figure 4–1 demonstrates the following scenario:

1. The client inserts record 1 and synchronizes the data to the server. The Mobile Server assigns the same magic number to both the client and the server to denote that they are in sync. In this example, the magic number on both the client and the server is 10.
2. The client inserts record 2. No synchronization is performed.
3. The backup is performed. Client record 1 is saved to the backup. The latest magic number on both the client and the server is 10.
4. The client inserts record 3 and synchronizes the data to the server. The Mobile Server assigns the same magic number to both the client and the server to denote that they are in sync. In this example, the magic number on both the client and the server is 20.
5. The client inserts record 4. No synchronization is performed.
6. A failure occurs and the last backup is used to recover the Mobile Server, the Mobile applications and the application data. In this scenario, only record 1 is in the backup, so it will exist in the restored database.
7. The client synchronizes. Records 2 and 3 can be lost, because they are not in the backup. The msync client does not send them to the server, since they were already sent in step 4. However, the msync client does send record 4 to the server, since it is a new record that has never been synchronized with the database. After the synchronization, record 4 is stored in the error queue, not in the application tables.

Figure 4–1 Lost Data With Backup and Recovery Strategy



The Mobile Server checks the magic numbers on both the client and the server. It verifies the state of the data on the client to determine what action to take. When the client performs the next synchronization, if the magic numbers are not the same, then the following occurs:

1. The client checks if there are any new records—whether newly inserted, modified, or deleted—on the client. If so, then these records are sent to the server, which saves these records in the error queue.
2. A full refresh of all of the subscribed data is sent to the client.

In our example, if you did nothing, the client would send record 4 to the server, which would end up in the error queue, and records 2 and 3 would be lost. To save records 2 and 3, do the following:.

1. On the server, retrieve and restore the last backup.
2. On the Mobile client that is out of sync, update any record that has been modified since the last synchronization. In our example, you would do any sort of update that makes the record seem to contain new information in records 2 and 3. For example, you could update the `VARCHAR` field with the same content.
3. Initiate a synchronization on the client. The Oracle Database Lite software detects that the client database is out of sync and that some of the records have been modified. Thus, the following occurs:
   a. The modified records are updated in the restored database on the server and saved in the error queue.
   b. The server pushes a full refresh down to the client.
4. In order for you to reapply the modified records to the applications table, you must first modify the DML operation from Error to Update. The DBA must modify the record in the error queue for the base table, named `CEQ$<base_table_name>`, changing the DML operation from Error (`E`) to Update (`U`) or Insert (`I`).
5. Once updated to Update, re-execute the command. Navigate to the Error Queue screen in the Mobile Manager. Click on the modified record. Click Execute.

> **Note:** For more information on the error queue and how to reapply the records, see Section 3.11.1 "Resolving Conflicts Using the Error Queue" in the *Oracle Database Lite Developer's Guide*.

6. The next time that the MGP runs, the update is applied to the application table.

Thus, all information contained within records 2 and 3 will be restored from the device.

4.4 Oracle Database Lite Recovery Issues

When you perform a recovery, the state of both the Mobile Server and the Mobile application schemas must be in-sync. If they are out-of-sync, severe problems may occur. Therefore, when you perform a backup and restoration for the Mobile Server and the Mobile application schemas, each must be recovered to the same point in time. Use the Oracle database Point-in-Time Recovery strategy to ensure that both the Mobile Server schema and Mobile application schemas are recovered to the same point in time.

The Mobile application schemas usually reside on the same Oracle database instance as the Mobile Server. However, if you have used a database link to store the Mobile application schemas on a separate Oracle database instance, then you must use a backup and restore strategy for distributed database systems.

Index

I

J

M

N

O

P

Q

R

S

T

U

V

W