CIS 363 MySQL

Chapter 21 Debugging MySQL Applications

Chapter 22 Basic Optimizations

Ch 21 Debugging MySQL Applications

- MySQL client programs produce diagnostic messages when they encounter problems. If problems occur when you attempt to connect to MySQL Server with a client program or while the server attempts to execute the SQL statements that you send to it, MySQL produces diagnostic messages.
- Diagnostics might be error messages to indicate serious problems or warning messages to indicate less severe problems. MySQL provides diagnostic information in the following ways:

Ch 21 Debugging MySQL Applications

- An error message is returned for statements that fail:

mysql> **SELECT * FROM no_such_table;**

ERROR 1146 (42S02) : Table 'test.no_such_table' doesn’t exist

MySQL-specific error code SQLSTATE error code A text that describes.

These message typically have three components:

- A MySQL-specific error code.
- An SQLSTATE error code. These codes are defined by standard SQL and ODBC.
- A Text message that describes the problem.

Ch 21 Debugging MySQL Applications

- ❖ An information string is returned by statements that affect multiple rows. This string provides a summary of the statement outcome:

```
mysql> INSERT INTO integers VALUES ('abc'), (-5), (null);
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warning: 3
```

- ❖ An operating system-level error might occur:

```
mysql> CREATE TABLE CountryCopy SELECT * FROM Country;
Error 1 (HY000): Can’t create/write to file './world.CountryCopy.frm'
(Errcode: 13)
```

Ch 21 Debugging MySQL Applications

- For case such as the preceding SELECT from a non0existent table, where all three error values are displayed, you can simply look at the information provided to see what the problem was. In other cases, all information might not be displayed. The information string for multiple-row statements is a summary, not a complete listing of diagnostics.

- An operating system error includes an Errcode number that might have a system-specific meaning.

- You can use the following means to obtain assistance in interpreting diagnostic information:
  - The **SHOW WARNINGS** and **SHOW ERRORS** statements display warning and error information for statements that produce diagnostic information.
  - The ***perror*** command-line utility displays the meaning of operating system-related error codes.
  - There is a chapter in the MySQL Reference Manual that lists error codes and messages.

Ch 21 Debugging MySQL Applications

The SHOW WARNINGS STATEMENT

- MySQL Server generates warnings when it is not able to fully comply with a request or when an action has possibly unintended side effects. These warnings can be displayed with the SHOW WARNINGS statement.

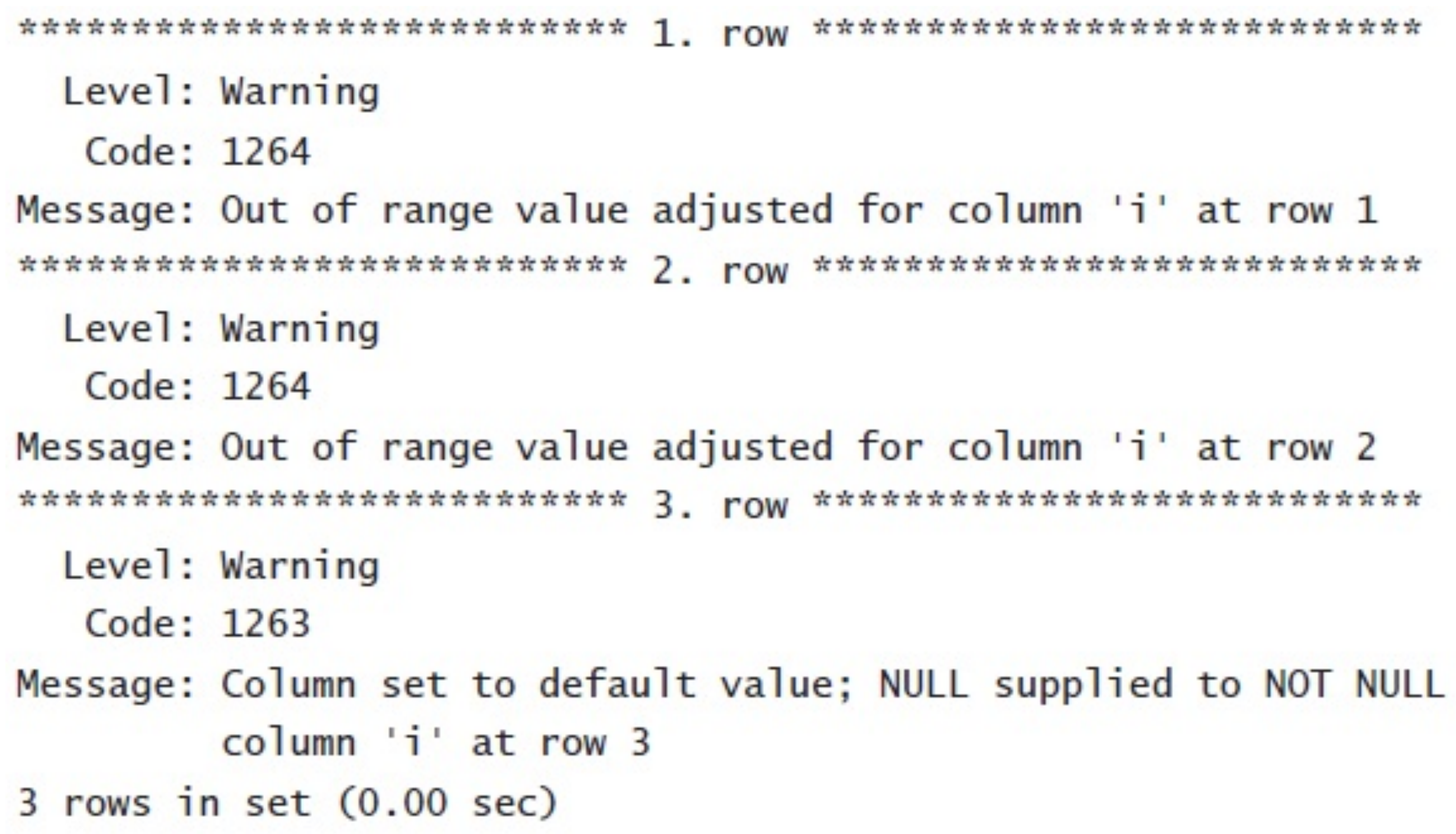
```
mysql> CREATE TABLE integers (I INT UNSIGNED NOT NULL);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO integers VALUES ('abc'), (-5), (NULL);
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warning: 3
```

- When a statement cannot be executed without some sort of problem occurring, the SHOW WARNINGS statement provides information to help you understand what went wrong.

Ch 21 Debugging MySQL Applications

mysql> SHOW WARNINGS\G

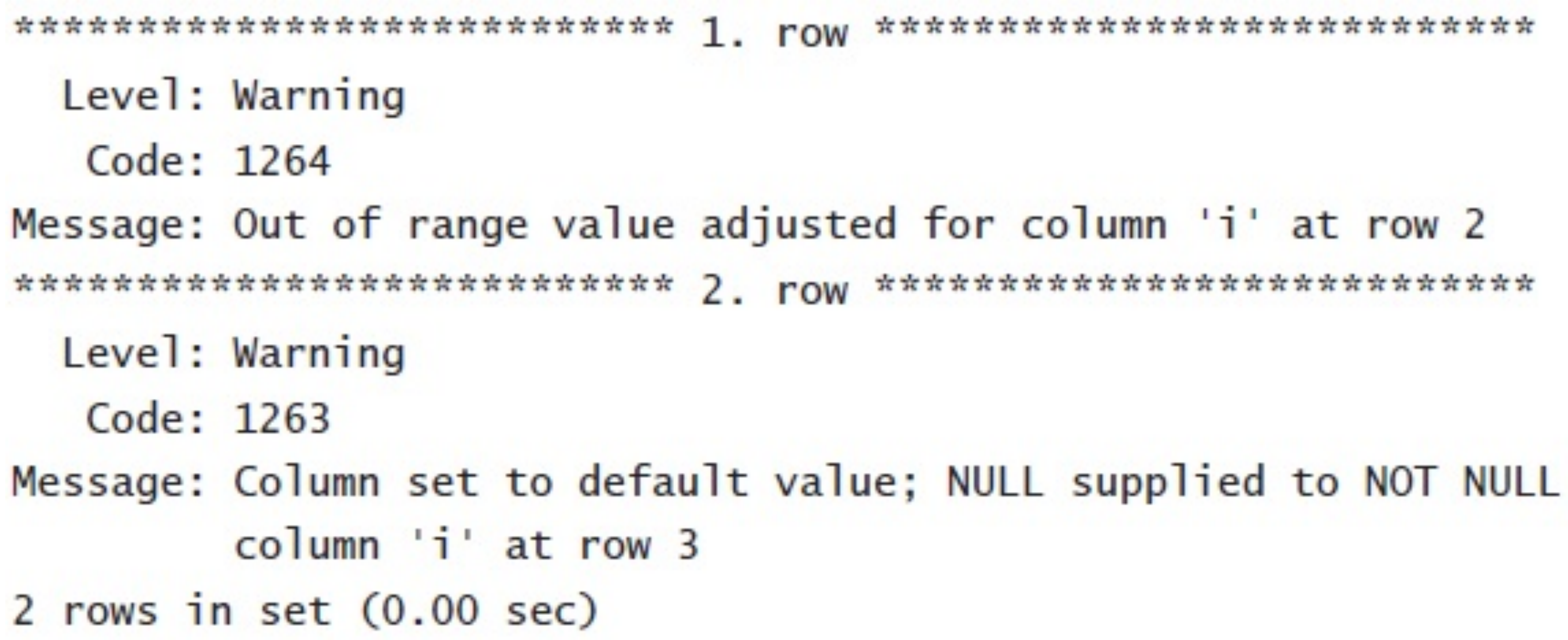


```
*************************** 1. row ***************************
  Level: Warning
   Code: 1264
Message: Out of range value adjusted for column 'i' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1264
Message: Out of range value adjusted for column 'i' at row 2
*************************** 3. row ***************************
  Level: Warning
   Code: 1263
Message: Column set to default value; NULL supplied to NOT NULL
         column 'i' at row 3
3 rows in set (0.00 sec)
```

Ch 21 Debugging MySQL Applications

SHOW WARNINGS with LIMIT

mysql> SHOW WARNINGS LIMIT 1,2\G



```
*************************** 1. row ***************************
  Level: Warning
   Code: 1264
Message: Out of range value adjusted for column 'i' at row 2
*************************** 2. row ***************************
  Level: Warning
   Code: 1263
Message: Column set to default value; NULL supplied to NOT NULL
         column 'i' at row 3
2 rows in set (0.00 sec)
```

Ch 21 Debugging MySQL Applications

To know how many warnings, use SHOW COUNT(*) WARNINGS.

```
mysql> SHOW COUNT(*) WARNINGS;
+-------------------------------------+
| @@session.warning_count |
+-------------------------------------+
|                  3                  |
+-------------------------------------+
```

- Warnings generated by one statement are available from the server only for a limited time. If you need to see warnings, you should always fetch them as soon as you defect that they were generated.

Ch 21 Debugging MySQL Applications

"Warning" actually can occur at several levels of severity:

- Error messages indicate serious problems that prevent the server from completing a request.
- Warning messages indicate problems for which the sever can continue processing the request.
- Note messages are informational only.

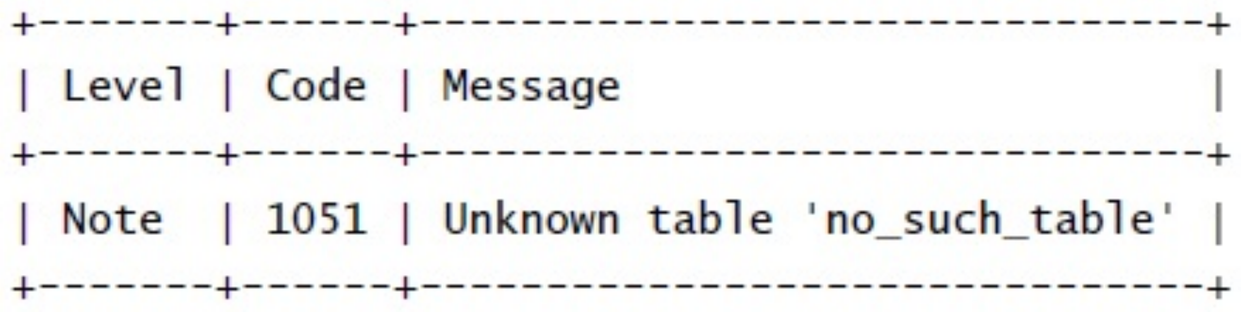
```
mysql> SELECT * FROM no_such_table;
ERROR 1146 (42S02) : Table  'test.no_such_table' doesn't exist
mysql> SHOW WARNINGS;
+-------+------+------------------------------------------+
| Level | Code | Message                                  |
+-------+------+------------------------------------------+
| Error | 1146 | Table 'test.no_such_table' doesn't exist |
+-------+------+------------------------------------------+
```

Ch. 21 Debugging MySQL Applications

mysql> **DROP TABLE IF EXISTS no_such_table;**

QUERY OK, 0 rows affected, 1 warning (0.00 sec)

mysql> **SHOW WARNINGS;**



```
+-------+------+-------------------------------+
| Level | Code | Message                       |
+-------+------+-------------------------------+
| Note  | 1051 | Unknown table 'no_such_table' |
+-------+------+-------------------------------+
```

- To suppress generation of Note warnings, you can set sql_notes system variable to zero:

mysql> SET sql_notes = 0;

Ch. 21 Debugging MySQL Applications

The SHOW ERRORS STATEMENT

- The SHOW ERRORS statement is similar to SHOW WARNINGS, but displays only messages for error conditions. As such, it shows only messages having a higher severity and tends to produce less output than SHOW WARNINGS.
- SHOW ERRORS, like SHOW WARNINGS, supports a LIMIT clause to restrict the number of rows to return. It also can be used as SHOW COUNT(*) ERROR to obtain a count of the error messages.

Ch. 21 Debugging MySQL Applications

The perror Utility

- Perror is a command-line utility that is included with MySQL distributions.
- The purpose of the perror program is to show you information about the error codes used by MySQL when operating system-level error occur. You can use perror in situations when a statement results in a message such as the following being returned to you.:

```
mysql> CREATE TABLE CountryCopy SELECT * FROM Country;
ERROR 1 (HY000): Can’t create/write to file './world.CountryCopy.frm'
```

- The error message indicates that MySQL cannot write to the file CountryCopy.frm, but does not report the reason. To find out, run the perror program with an argument of the number given following Errcode in the preceding error message.

```
Shell> perror 13
Error code 13: Permission denied.
```

Ch. 22 Basic Optimizations

Overview of Optimization Principles

There are several optimization strategies that you can take advantage of to make your queries run fast.:

- The primary optimization technique for reducing lookup times is to use indexing properly. This is true for retrievals (SELECT statement), and indexing also reduces row lookup time for UPDATE and DELETE statements as well.
- The way a query is written might prevent indexes from being used even if they are available. Rewriting the query often will alow the optimizer to use an index and process a query faster.
- The EXPLAIN statement provides information about how the MySQL optimizer processes queries.
- In some cases, query processing for a task can be improved by using a different approach to the problem such as generating summary tables rather than selecting from the raw data repeatedly.
- Queries run more efficiently when you choose a storage engine with properties that best match application requirements.

Ch 22 Basic Optimizations

- The optimization is important for MySQL because it reduces query execution time and it helps everyone who uses the server. When the server runs more smoothly and processes more queries with less work, it performs better as a whole.
- A query that takes less time to run doesn’t hold locks as long, so other clients that are trying to update a table don’t have to wait so long. This reduces the chance of a query backlog building up.
- A query might be slow due to lack of proper indexing. If MySQL cannot find a suitable index to use, it must scan a table in its entirety. For a large table, that involves a lots of processing and disk activity. This extra overhead affects not only your own query, it takes machine resources that could be devoted to processing other queries. Adding effective indexes allows MySQL to read only the relevant parts of the table, which is quicker and less disk intensive.

Ch 22 Basic Optimizations

Using Indexes for Optimization

When you create a table, consider whether it should have indexes, because they have important benefits:

- Indexes contain sorted values. This allows MySQL to find rows containing particular values faster. The effect can be particularly dramatic for joins.
- Indexes result in less disk I/O. The sever can use an index to go directly to the relevant table records. Furthermore, if a query displays information only from indexed columns, MySQL might be able to process it by reading only the indexes and without accessing data rows at all.

MySQL supports several types of indexes:

A primary Key: Every value must be non-NULL

A UNIQE index: similar to Primary Key but it can be defined to allow NULL values

A Non-Unique index: This type of index is defined with the keyword INDEX or KEY.

A FULLTEXT: specially designed for text searching. (in MyISAM tables)

A SPATIAL: be used with the spatial data type. (not covered on MySQL cert exam)

Ch 22 Basic Optimizations

- An Index helps MYSQL perform retrievals more quickly than if no index is used, but indexes can be used with varying degrees of success. Keep the following index-related considerations in mind when designing tables:
  - Declare an indexed column NOT NULL if possible. Although NULL values can be indexed, NULL is a special value that requires additional decisions by the server when performing comparisons on key values.
  - Avoid over indexing. Unnecessary indexing can slows down table updates.
  - An index on a column that has very few distinct values is unlikely to do much good such as ENUM or SET data types.
  - Index a column prefix rather than the entire column.Shortening the length of key values can improve performance by reducing the amount of disk I/O needed to read the index and by increasing the number of key values that fit into the key cache.
  - Avoid creating multiple indexes that overlap.
  - Use ALTER TALBE to add indexes in the same statement. Avoid using CREATE INDEX because it allows only one index to be added or dropped.

Ch 22 Basic Optimizations

Indexing Column Prefixes

```
mysql> CREATE TABLE t (name CHAR(225), INDEX (NAME));
```

If you index all 255 characters of the values in the name column, index processing will be relatively slow:

- It’s necessary to read more information form disk
- Longer values take longer to compare.
- The index is not as effective because fewer key values fit into it at a time.

To overcome this problem, you can index only prefix of the column values.

mysql> CREATE TABLE t (name CHAR(225), INDEX (NAME**(15)**));

- To specify a prefix length for a column, follow the column name in the index definition by a number in parentheses.

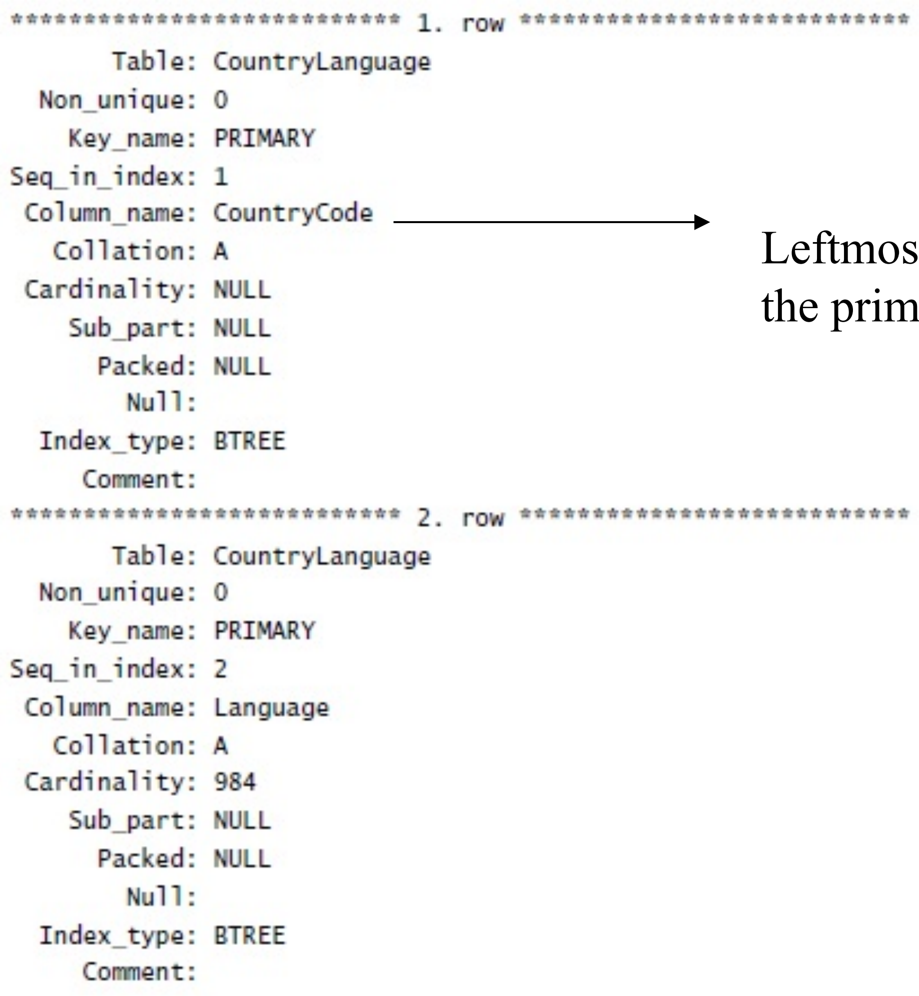
Ch 22 Basic Optimizations

Leftmost Index Prefixes

- In a table that has a composite (multiple-column) index, MySQL can use leftmost index prefixes of that index. A leftmost prefix of a composite index consists of one or more of the initial columns of the index. MySQL's capability to use leftmost index prefixes enables you to avoid creating unnecessary (redundant) indexes.
- Note that a leftmost prefix of an index and an index on a column prefix are two different things. A leftmost prefix of an index consists of leading columns in a multiple-column index. An index on a column prefix indexes the leading characters of values in the column.

```
mysql> SHOW INDEX FROM CountryLanguage\G
```

Ch 22 Basic Optimizations



```
*************************** 1. row ***************************
       Table: CountryLanguage
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 1
 Column_name: CountryCode
   Collation: A
 Cardinality: NULL
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
*************************** 2. row ***************************
       Table: CountryLanguage
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 2
 Column_name: Language
   Collation: A
 Cardinality: 984
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
```

$\longrightarrow$ Leftmost prefix of the primary key

Ch 22 Basic Optimizations

General Query Enhancement

There are some general techniques to improve your query performance:

- Writing queries in a more efficient way.
- Using EXPLAIN to obtain optimizer information
- Optimizing queries by limiting output
- Using summary tables
- Optimizing updates

Ch 22 Basic Optimizations

Query Rewriting Techniques

- The way you write a query often affects who well indexes are used. Use the following principles to make your queries more efficient:
  - Don't refer ti an indexed column within an expression that must be evaluated for every row in the table. Doing so prevents use of the index.

Suppose that a table t contains a DATE column d that is indexed.

```
mysql> SELECT * FROM t WHERE YEAR >=1994;
mysq> SELECT * FROM t WHERE d >= '1994-01-01'
```

$\longrightarrow$ Better. The index can be used.

  - Indexes are particular beneficial for joins that compare columns from two tables.

```
mysql> SELECT * FROM Country, CountryLanguage WHERE Country.Code = CountryLanguage.CountryCode;
```

*** If neither the code nore CountryCode column is indexed, every par of column values must be compared to find those pairs that are equal.

Ch 22 Basic Optimizations

- When comparing an indexed column to a value, use a value that has the same data type as the column. If you are look for rows containing a numeric id value of 18 with either of the following WHERE clause: (Both produce the same result)

  WHERE id = 18 ⟶ Better

  WHERE is = '18' ⟶ For string value, MySQL must perform a string-to-number conversion.

- In certain cases, MySQL can use an index for pattern-matching operations performed with the LIKE operator. This is true if the pattern begins with a literal prefix value rather with a wildcard character. An index on a name column can be used for a pattern match like this:

  WHERE name LIKE 'de%'

  On the other hand, the following pattern makes LIKE more difficult for the optimizer:

  WHERE name LIKE '%de%'

  When a pattern starts with a wildcard character as just shown, MySQL cannot make efficient use of any indexes associated with the column.

Ch 22 Basic Optimizations

Using EXPLAIN to obtain optimizer Information

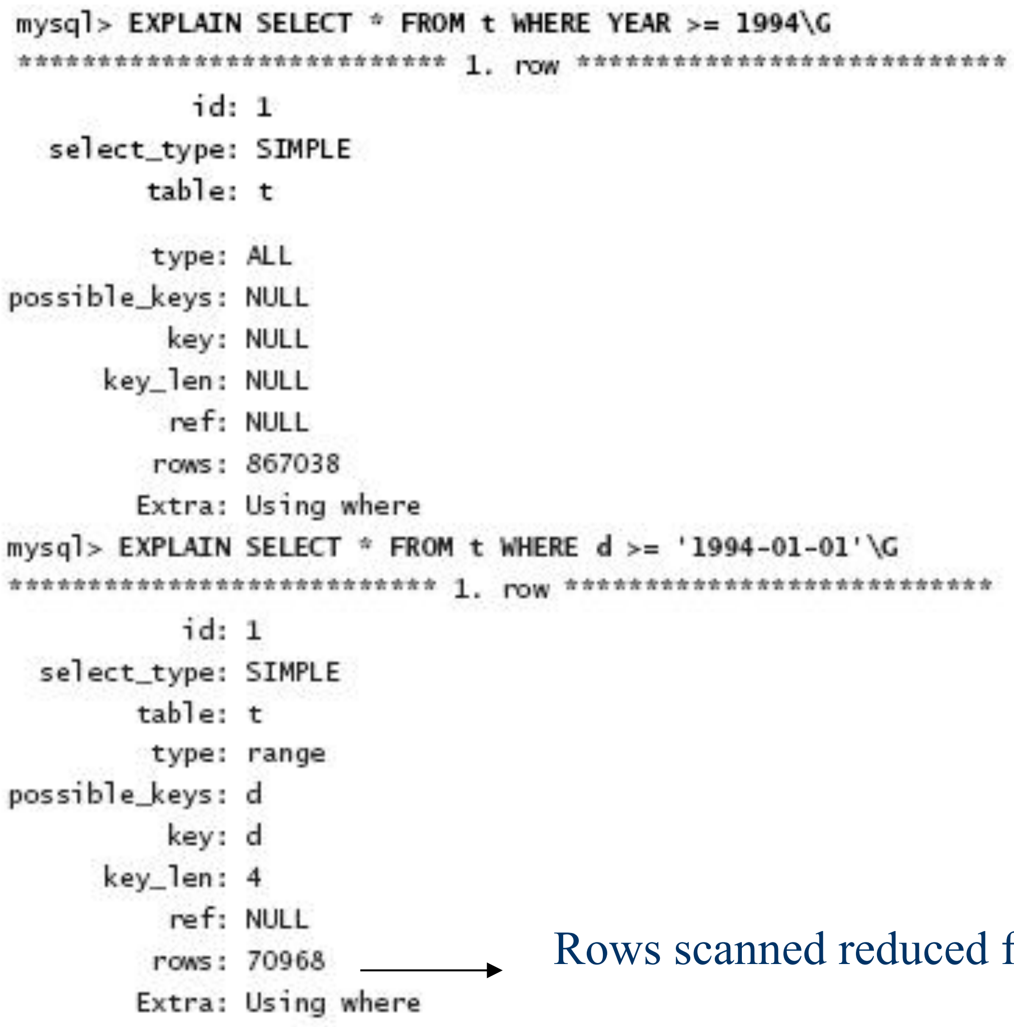
```
mysql> SELECT * FROM t WHERE YEAR >=1994;
mysq> SELECT * FROM t WHERE d >= '1994-01-01'
```

$\longrightarrow$ Why this one is more efficient? Use EXPLAIN

To verify whether MySQL actually will use an index to process the second query, use the EXPLAIN statement to get information from optimizer about the execution plans it would use.

mysql> EXPLAIN SELECT * FROM t WHERE YEAR >=1994\G

Ch 22 Basic Optimizations



```
mysql> EXPLAIN SELECT * FROM t WHERE YEAR >= 1994\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 867038
        Extra: Using where
mysql> EXPLAIN SELECT * FROM t WHERE d >= '1994-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
         type: range
possible_keys: d
          key: d
      key_len: 4
          ref: NULL
         rows: 70968
        Extra: Using where
```

$\longrightarrow$ Rows scanned reduced from 867038 to 70968.

Ch 22 Basic Optimizations

Optimizing Queries by Limiting Output

- Some optimizations can be done independently on whether indexes are used. One way to eliminate unnecessary output is by using a LIMIT clause. This helps in two ways:
  - Less information need be returned over the network to the client.
  - LIMIT allows the server to terminate query processing earlier than it would otherwise. Some row-sorting techniques have the property that the first ***n*** rows can be known to be in the final order even before the sort has been done completely. This means that when LIMIT ***n*** is combined with ORDER BY, the server might be able to determine the first ***n*** row and then terminate the sort operation early.
- Don’t use LIMIT to pull out a few rows from a gigantic result set. Instead, try to use a WHERE clause that restricts the result so that the server doesn’t retrieve as many rows in the first place.
- Another way to reduce query output is to limit it “horizontally”. It means select the columns you need, rather than using SELECT * to retrieve all columns.

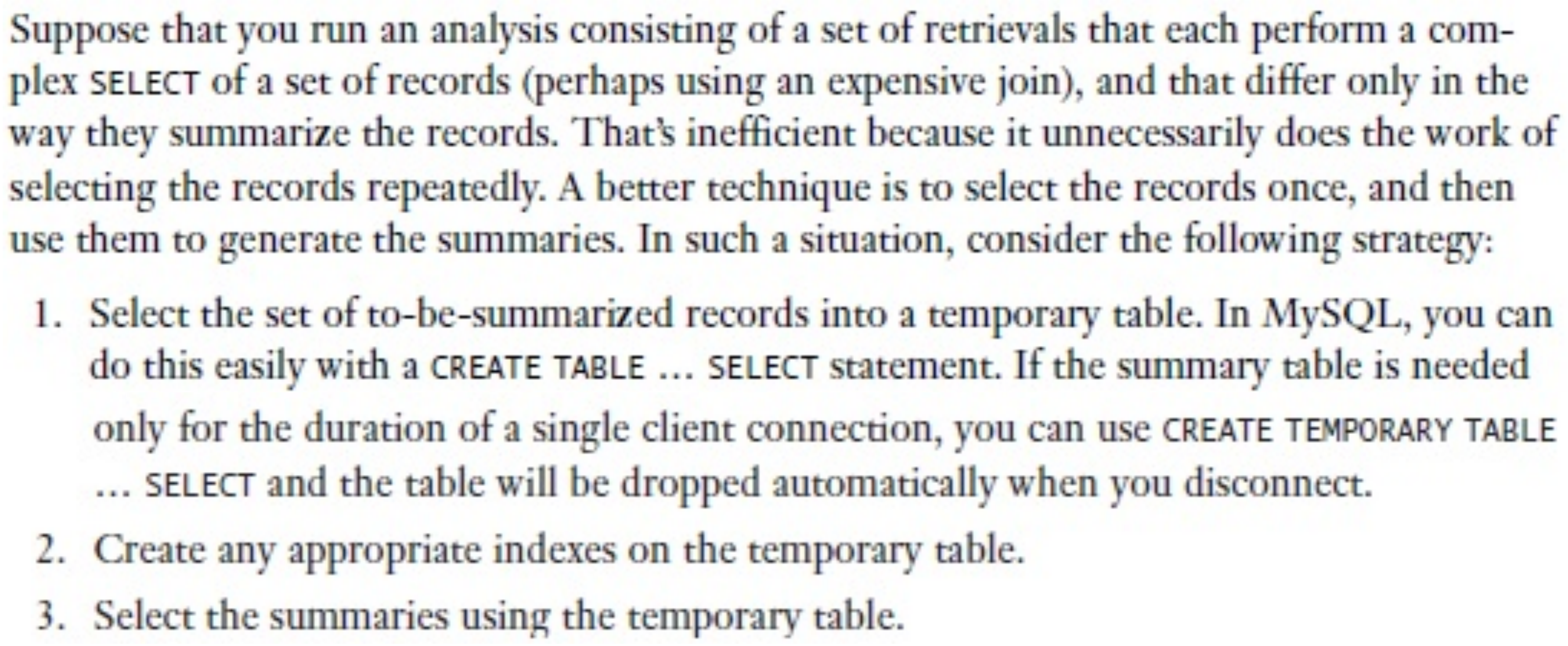
Ch 22 Basic Optimizations

```
mysql> SELECT * FROM Country WHERE Name LIKE 'M%';
mysql> SELECT name FROM Country WHERE Name LIKE 'M%';
```
$\longrightarrow$ Better

- The second query is faster because MySQL has to return less information when you select just one column rather than all of them.
- In addition, if an index on NAME exists, you can get even more improvement:
  - The index can be used to determine quickly which Name values satisfy the condition in the WHERE clause. This is faster than scanning the entire table.
  - Depending on the store engine, the server might not read the table rows at all. If the values requested by the query are in the index, then by reading the index MYSQL already has the information that the client requested. For example, the MyISAM engine can read the index file to determine which values satisfy the query, and then return them to the client without reading the data file at all. Doing so is faster than reading both the index file and the data file.

Ch 22 Basic Optimizations

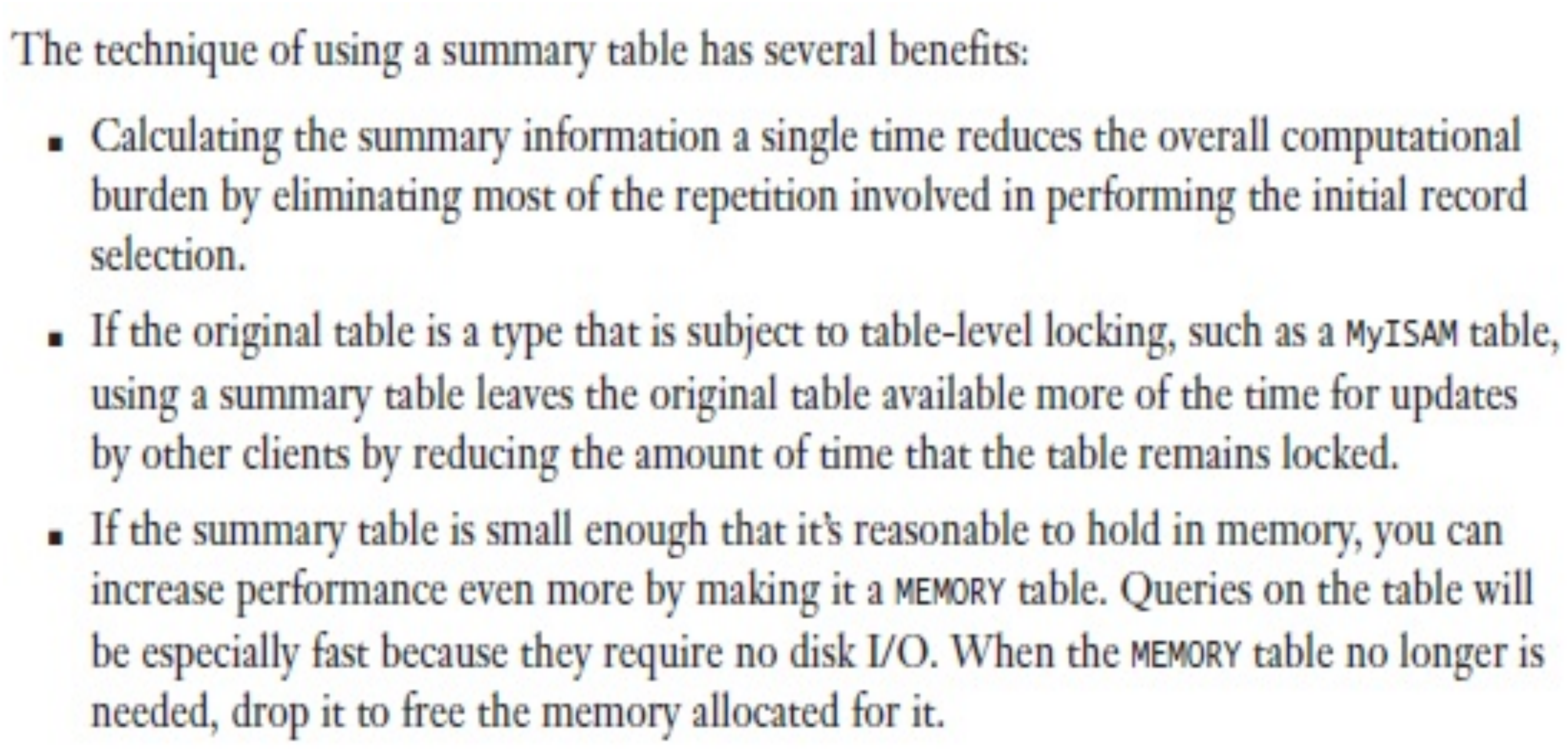
Using Summary Tables



Suppose that you run an analysis consisting of a set of retrievals that each perform a complex SELECT of a set of records (perhaps using an expensive join), and that differ only in the way they summarize the records. That's inefficient because it unnecessarily does the work of selecting the records repeatedly. A better technique is to select the records once, and then use them to generate the summaries. In such a situation, consider the following strategy:

1. Select the set of to-be-summarized records into a temporary table. In MySQL, you can do this easily with a CREATE TABLE … SELECT statement. If the summary table is needed only for the duration of a single client connection, you can use CREATE TEMPORARY TABLE … SELECT and the table will be dropped automatically when you disconnect.
2. Create any appropriate indexes on the temporary table.
3. Select the summaries using the temporary table.

Ch 22 Basic Optimizations



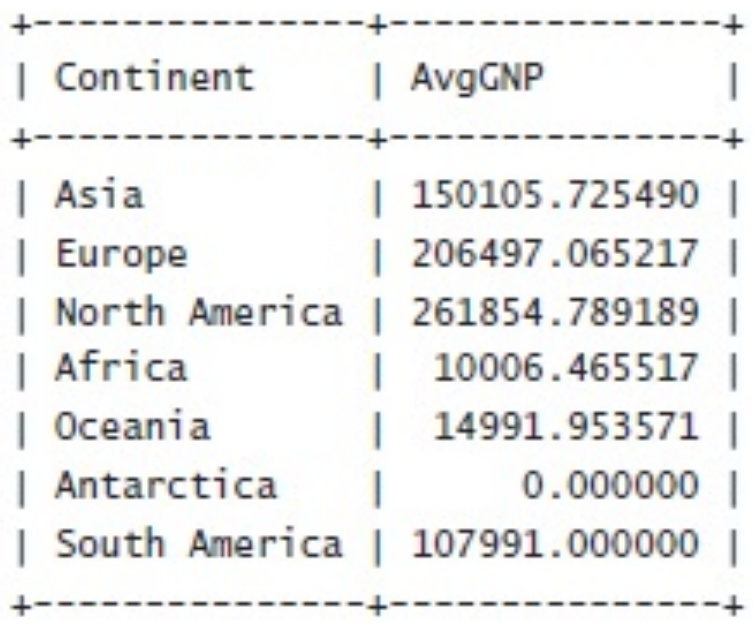
The technique of using a summary table has several benefits:

- Calculating the summary information a single time reduces the overall computational burden by eliminating most of the repetition involved in performing the initial record selection.
- If the original table is a type that is subject to table-level locking, such as a `MyISAM` table, using a summary table leaves the original table available more of the time for updates by other clients by reducing the amount of time that the table remains locked.
- If the summary table is small enough that it's reasonable to hold in memory, you can increase performance even more by making it a `MEMORY` table. Queries on the table will be especially fast because they require no disk I/O. When the `MEMORY` table no longer is needed, drop it to free the memory allocated for it.

Ch 22 Basic Optimizations

Example: (creating a summary table containing the average GNP value of countries in each continent)

```
mysql> CREATE TABLE ContinentGNP SELECT Continent, AVG(GNP) AS AvgGNP FROM
        Country GROUP BY Continent;
mysql> SELECT * FROM ContinentGNP;
```

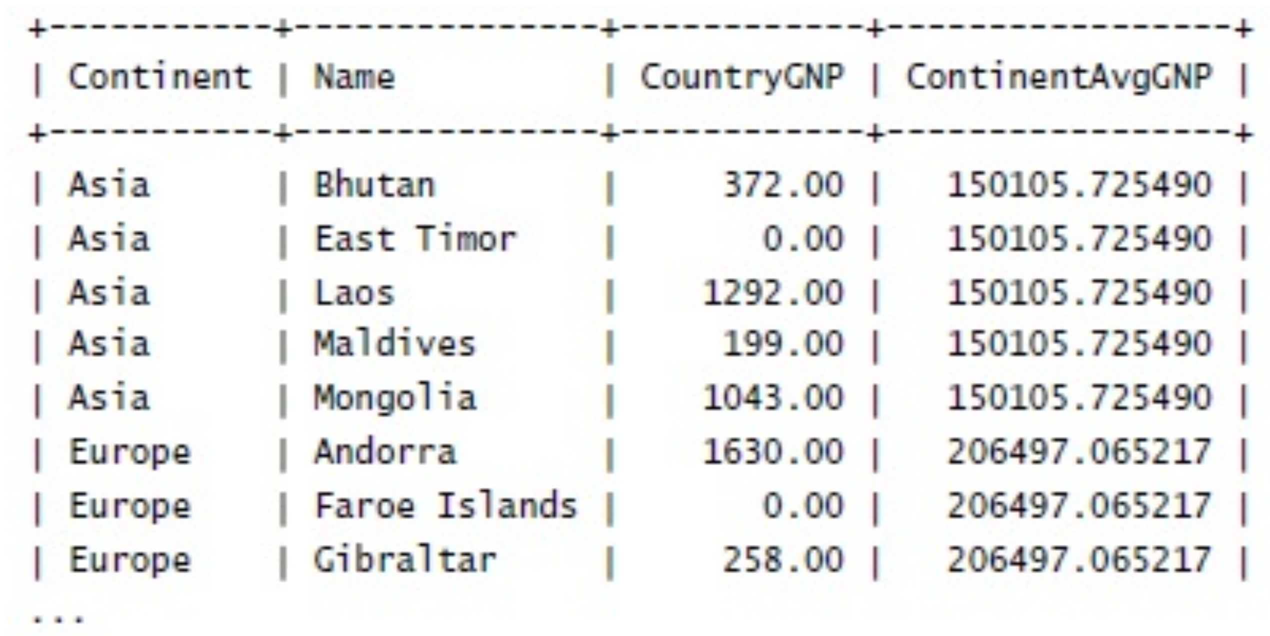


```
+---------------+---------------+
| Continent     | AvgGNP        |
+---------------+---------------+
| Asia          | 150105.725490 |
| Europe        | 206497.065217 |
| North America | 261854.789189 |
| Africa        |  10006.465517 |
| Oceania       |  14991.953571 |
| Antarctica    |      0.000000 |
| South America | 107991.000000 |
+---------------+---------------+
```

Ch 22 Basic Optimizations

- Compare the summary table to the original table to find countries that have a GNP less than 1% of the continental average:

mysql> SELECT Country.Continent, Country.Name, Country.GNP AS CountryGNP, ContinentGNP.AvgGNP AS ContinentAvgGNP FROM Country, ContinentGNP WHERE Country.Continent = (ContinentGNP.Continent) AND Country.GNP < ContinentGNP.AvgGNP * .01 ORDER BY Country.Continent, Country.Name;

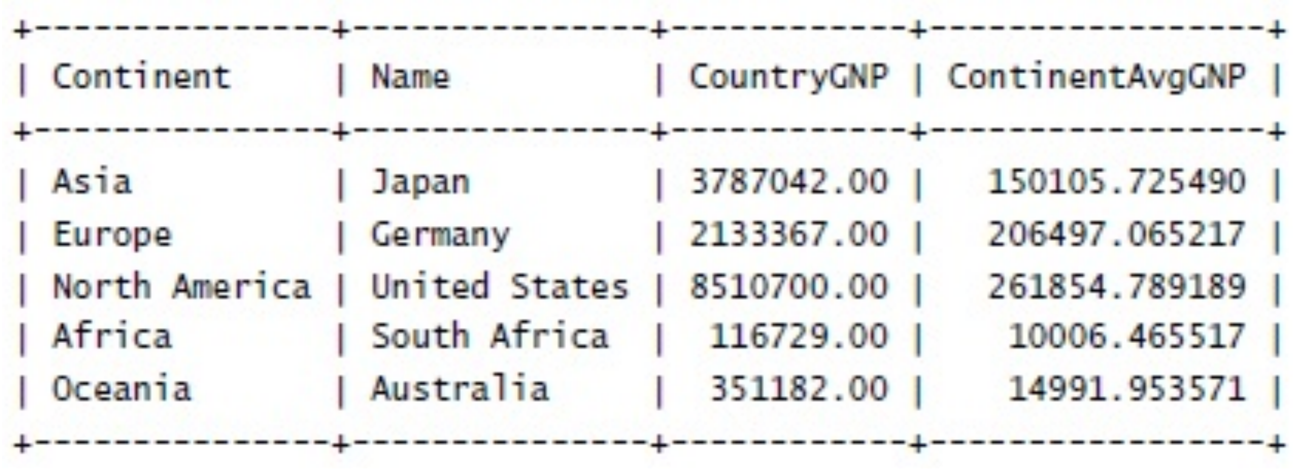


```
+-----------+---------------+------------+-----------------+
| Continent | Name          | CountryGNP | ContinentAvgGNP |
+-----------+---------------+------------+-----------------+
| Asia      | Bhutan        |     372.00 |   150105.725490 |
| Asia      | East Timor    |       0.00 |   150105.725490 |
| Asia      | Laos          |    1292.00 |   150105.725490 |
| Asia      | Maldives      |     199.00 |   150105.725490 |
| Asia      | Mongolia      |    1043.00 |   150105.725490 |
| Europe    | Andorra       |    1630.00 |   206497.065217 |
| Europe    | Faroe Islands |       0.00 |   206497.065217 |
| Europe    | Gibraltar     |     258.00 |   206497.065217 |
...
```

Ch 22 Basic Optimizations

- Use the summary table to find countries that have a GNP more than 10 times the continental average.

```
mysql> SELECT Country.Continent, Country.Name, Country.GNP AS CountryGNP,
    ContinentGNP.AvgGNP AS ContinentAvgGNP FROM Country, ContinentGNP WHERE
    Country.Continent = ContinentGNP.Continent AND Country.GNP > ContinentGNP.AvgGNP * 10
    ORDER BY Country.Continent, Country.Name;
```



```
+---------------+---------------+------------+-----------------+
| Continent     | Name          | CountryGNP | ContinentAvgGNP |
+---------------+---------------+------------+-----------------+
| Asia          | Japan         | 3787042.00 |   150105.725490 |
| Europe        | Germany       | 2133367.00 |   206497.065217 |
| North America | United States | 8510700.00 |   261854.789189 |
| Africa        | South Africa  |  116729.00 |    10006.465517 |
| Oceania       | Australia     |  351182.00 |    14991.953571 |
+---------------+---------------+------------+-----------------+
```

Ch 22 Basic Optimizations

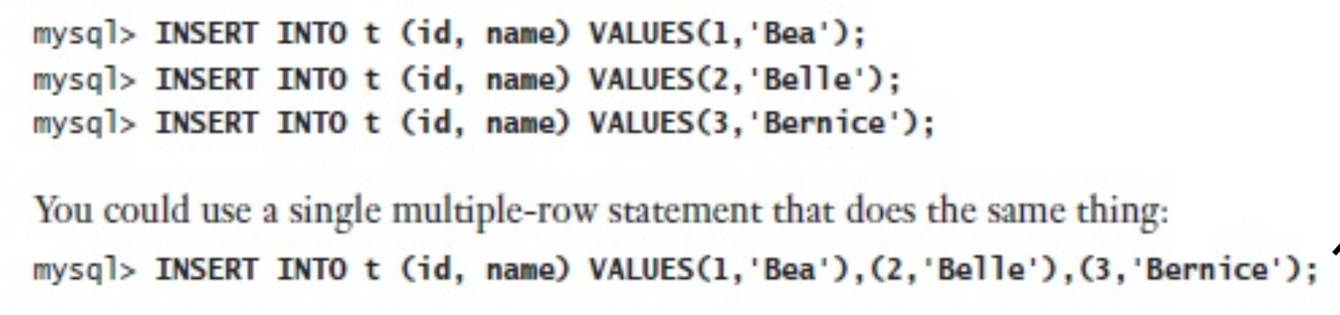
- Use of summary tables has the disadvantage that the records they contain are up to date as long as the original values remain unchanged. If the original table rarely or never changes, this might be only a minor concern.
- For many applications, summaries that are close approximations are sufficiently accurate.
- The summary table technique can be applied at multiple levels. Create a summary table that holds the results of an intial summary, and then summarize that table in different ways to produce secondary summaries. This aviods the computational expense of generating the initial summary repeatedly.

Ch 22 Basic Optimizations

Optimizing Updates

Optimization techniques can be used for statements that update tables.

- For a DELETE or UPDATE statement that uses a WHERE clause, try to write it in a way that allows an index to be used for determining which rows to delete or update.
- EXPLAIN is used with SELECT queries, but you might also find it helpful for analyzing UPDATE and DELETE statements. Write a SELECT that has the same WHERE clauses as the UPDATE or DELETE and analyze that.
- USE multiple-row INSERT statement instead of of multiple single-row INSERT statements.



```
mysql> INSERT INTO t (id, name) VALUES(1,'Bea');
mysql> INSERT INTO t (id, name) VALUES(2,'Belle');
mysql> INSERT INTO t (id, name) VALUES(3,'Bernice');
```

You could use a single multiple-row statement that does the same thing:

```
mysql> INSERT INTO t (id, name) VALUES(1,'Bea'),(2,'Belle'),(3,'Bernice');
```

Query is shorter. Less information to send to the server. It allows the server to perform all the updates at once and flush the index a single time. This optimization can be used with any storage engine.

Ch 22 Basic Optimizations

- If you’re using an InnoDB table, you can get better performance even for single-row statements by grouping them within a transaction rather than by executing them with autocommit mode enabled:

```
mysql> START TRANSACTION;
mysql> INSERT INTO t (id, name) VALUES(1, 'Bea')
mysql> INSERT INTO t (id, name) VALUES(2, 'Belle')
mysql> INSERT INTO t (id, name) VALUES(3, 'Bernice')
mysql> COMMIT;
```

- Using a transaction allows InnoDB to flush all the changes at commit time. In autocommit mode, InnoDB flushes the changes for each INSERT individually.
- For any storeage engine, LOAD DATA INFILE is even faster than multiple-row INSERT statements.
- You can disable index updating when loading data into an empty MyISAM table to speed up the operation. LOAD DATA INFILE does this automatically for non-unique indexes of the table is empty.
- To replace existing rows, use REPLACE rather than DELETE plus INSERT.

Ch 22 Basic Optimizations

Choosing Appropriate Stroage Engine

- It is important to choose a storage engine that uses a locking level appropriate for the anticipated query mix when you create a table.
  - MyISAM table-level locking works best for a query mix that is heavily skewed toward retrievals and includes few updates.
  - Use InnoDB if you must process a query mix containing many updates. InnoDB's use of row-level locking and multi-versioning providing good concurrency for a mix of retrievals and updates.
  - Different MyISAM storage formats have different performance characteristics. This influences whether you choose fixed-length or variable-length columns to store string data.
    - Use fixed-length column (CHAR, BINARY) for best speed
    - USE variable-length columns (VARCHAR, VARBINARY, TEXT, BLOB) for best use of disk space.
  - Another option with MyISAM tables is to use compressed read-only tables.

Ch 22 Basic Optimizations

- For InnoDB tables, it is true that CHAR columns take more space on average than VARCHAR. There is no retrieval speed advantage for InnoDB as there is with MyISAM, because the InnoDB engine implements storage for both CHAR and VARCHAR in a similar way. Retrieval of CHAR values might be slower because on average they require more information to be read from disk.
- MERGE tables can use a mix of compressed and uncompressed tables. This can be useful for time-based records. For example, if you lo records each year to a different log file, you can use an uncompressed log table for the current year so that you can updated it, but compress the tables for past years to save space. If you then create a MERGE table from the collection, you can easily run queries that search all tables together.

(More information about storage engine-specific optimizations can be found in chapter 38.)