A Case Study of eBay UTF-8 Database Migration

Nelson Ng
Chief Globalization Architect
What is the difficulty in migrating from ISO Latin 1 to UTF-8?
An online trading platform where everyone can trade almost anything from anywhere

- A global presence in 33 international markets
- 203 million registered users
- $44.3 billion in annualized gross merchandise volume (value of goods sold)
- 724,000 small business and individuals in U.S. use eBay as a primary or secondary sales channel
- 36,000 Developers via eBay API

Data as of Q2 2006.
Agenda

• Reasons for UTF-8 Migration
• Challenges of UTF-8 Migration
• Approaches to UTF-8 Migration
• Lessons Learned
Reasons for UTF-8 Migration
## Ambiguity in Textual Data Representation

### Raw Byte Sequence in Hex representation

<table>
<thead>
<tr>
<th></th>
<th>63</th>
<th></th>
<th>61</th>
<th></th>
<th>C3</th>
<th>91</th>
<th>E4</th>
<th>B8</th>
<th>AD</th>
<th>E5</th>
<th>9C</th>
<th>8B</th>
<th>64</th>
<th></th>
<th>C3</th>
<th>81</th>
<th>E2</th>
<th>82</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ISO 8859-1

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>a</th>
<th>Á</th>
<th>?</th>
<th>ä</th>
<th>,</th>
<th>-</th>
</tr>
</thead>
</table>

### CP1252

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>a</th>
<th>Á</th>
<th>‘</th>
<th>ä</th>
<th>,</th>
<th>-</th>
<th>å</th>
<th>œ</th>
<th>&lt;</th>
</tr>
</thead>
</table>

### UTF-8

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>a</th>
<th>Ñ</th>
<th>中</th>
<th>d</th>
<th>Á</th>
<th>€</th>
</tr>
</thead>
</table>

---

Raw Byte Sequence in Hex representation:

- 63
- 61
- C3 91
- E4 B8 AD
- E5 9C 8B
- 64
- C3 81
- E2 82 AC

Translation:

- ISO 8859-1: `c a Á`  
- CP1252: `c a å`  
- UTF-8: `c a Ñ`
eBay Marketplace supports both Latin 1 and UTF-8 sites on a single platform:

- **Latin 1 Sites:** US, CA, UK, AU, FR, DE, IT, BE (nl), BE (fr), NL, ES, CH, AT, IE, NZ
- **UTF-8 Sites:** TW, IN, CN, HK, MY, PH, PL, SG, SE

- Textual data for all sites are stored in the same set of tables and DB hosts

- A combination of ISO Latin 1 and UTF-8 DB hosts
  - **ISO Latin 1 DB host:**
    - **Latin 1 Site records:** Encoded in Cp1252
    - **UTF-8 Site records:** Encoded in UTF-8
  - **UTF-8 DB host:**
    - **All Site records:** Encoded in UTF-8
Textual Data Encoding Flow

Latin 1 Site Web Page

UTF-8 Site Web Page

Latin 1 Site Web Page

UTF-8 Site Web Page

Internal Textual Data Encoding Flow

Java String

CP 1252

UTF-8

Byte Seq

Byte Seq

ISO 8859-1

ISO 8859-1

Java String

Site DB

WE8ISO8859P1

Site ID

Textual Data

US
Cp1252

CN
UTF-8

Site ID

Textual Data

US
UTF-8

CN
UTF-8
Issues with Textual Data Encoding Model

• Cross-Border Trade Impediments

• Increased code complexity and developmental costs to support mixed textual data encoding model

• Increased 3rd Party integration costs to implement eBay textual data encoding model

• Textual data encoding model not supported by Oracle
Challenges of UTF-8 Migration
Summary of Challenges

- **Stringent Availability**
  - Site Uptime Requirement: 99.94%

- **Magnitude of Data**
  - Total Site DB Hosts: ~160
  - Total Active DB Tables: ~2000
  - Total DB Storage Usage: ~200TB
  - Total Redo Log Generation: ~2TB/day

- **Extreme Usage**
  - Peak time SQL Execution: ~462K/sec
  - Avg. time for SELECT: ~7ms
  - Avg. time for INSERT/UPDATE: ~15ms
  - Total Application Servers: ~5K

- **Frequency of Change**
  - Release Cycle: Every two weeks

*Notes: DB related metrics are for Primary site facing DB only. This doesn't including non site facing functions such as DR, Archive, DW etc.*
Approach to UTF-8 Migration
How Big was the Problem?

- Scanning DB hosts with Oracle CSSCAN
- Identify different types of textual data encoding
  - 7 bit ASCII (0x0 to 0x7F): No conversion. Same value as-is in UTF-8
  - 8 bit Cp1252 (0x80 to 0xFF): Converted into 2 bytes or 3 bytes for UTF-8.
  - UTF-8 (2 bytes or 3 bytes): No conversion. Same value as-is for UTF-8

* Data was gathered in Oct 2003
Approaches

1. Conversion with External File

2. Conversion with Secondary DB

3. In-line Conversion WITHOUT Status flag

4. In-line Conversion WITH Status flag
Stages of UTF-8 DB Migration

Pre Conversion

- Development & DB Preparation
  - Identify columns to be migrated
  - Expand column width where necessary
  - Identify CLOB columns
  - Add UTF-8 Status flag
  - Add flag to activate Implicit Conversion

Cp1252 & UTF-8 Data in ISO Latin 1 DB

Code Deployment

Implicit Conversion

Continue to generate new data in Cp1252 & UTF-8 data

Explicit Conversion

Generating new data in UTF-8

DB Migration

ALTER DB Charset

Converting existing data to UTF-8

Completed Migration

DB offline

Convert CLOB

All new data in UTF-8

UTF-8 Data in ISO Latin 1 DB

UTF-8 Data in UTF-8 DB
Pre Conversion and Data Conversion Stages

1. Create a UTF-8 Status column with default value NULL in each table.
2. Modify code base to support UTF-8 Migration to check DB Charset
   a. If DB is UTF-8, then use UTF-8 rules to process textual data for all sites
   b. If DB is ISO Latin 1, then process textual data according to the UTF-8 Status column
3. Activate Implicit Conversion
4. Run conversion script for Explicit Conversion.
DB Migration Stage

1. Stop Incoming Traffic to DB Host

2. ALTER DATABASE CHARSET from \texttt{WE8ISO8859P1} to \texttt{UTF8}

3. Restart Incoming Traffic to DB Host
Reasons for this Approach

Pros

• No DB downtime during data conversion. Only minimal DB outage during ALTER DB

• Implicit Conversion to reduce Explicit Conversion effort

• Ability to pause Data Conversion if necessary

• Ability to rollback ALTER DB if necessary

• Does not require additional hardware for secondary DB

Cons

• A new UTF-8 Status column has to be created for each table

• All DB access code has to be modified to handle the new UTF-8 Status column during transition period of data conversion
Lessons Learned
Lessons Learned

• Planning! Planning! Planning!
  ➢ Upfront Data Analysis
  ➢ DB Environment Preparation
  ➢ Leverage Site Maintenance Windows

• Gradual Activation of Data Conversion

• Data Monitoring
  ➢ Data Environment Changes
  ➢ Integrity of Data Conversion
  ➢ Problematic SQL in DB Log

• Throttle Data Conversion Speed

• Dedicated support resources from Development and Operations teams
Metrics of Migration

Duration of Explicit Conversion: ~7 months
- Number of Records: ~18 billion
- Number of Columns: ~600
- Number of Tables: ~200

Duration of DB Migration: ~7 months
- Number of DB Host: ~100
- Site Uptime for Migration Period: > 99.94%
Q & A
Appendix
Alternative 1: Conversion with External File

1. Identify all records with 8 bit Cp1252(0x80 to 0xFF) textual data
2. Extract identified records into a file
3. Use a script to convert the textual data in the file to UTF-8
4. ALTER DATABASE CHARSET from WE8ISO8859P1 to UTF8
5. Reinsert all extracted records back into DB
Alternative 1: Conversion with External File

**Pros**

• Avoids creating UTF-8 Status column in each table

• Code change to handle transition period during data migration is not required.

• Avoids requiring additional hardware for secondary DB

**Cons**

• Data conversion must be completed in a single outage window

• Prolong DB downtime due to data conversion takes place during outage window
Alternative 2: Conversion with Secondary DB

1. Create Last Modified Date (LMD) column in each table
2. Replicate Primary DB to Secondary DB and save the timestamp
3. In Secondary DB, identify all records with 8 bit Cp1252 (0x80 to 0xFF) textual data
4. In Secondary DB, use a script to convert the identified records to UTF-8
5. In Primary DB, run post conversion script to sync up modified records since DB replication
6. In Secondary DB, ALTER DB CHARSET from WE8ISO8859P1 to UTF8
7. Redirect site traffic to Secondary DB
Alternative 2: Conversion with Secondary DB

Pros

• Avoids creating UTF-8 Status column in each table

• Code change to handle transition period during data migration is not required.

• Data conversion can be restarted if necessary

• No data quality impact to source during data conversion

• No DB outage for ALTER DB. Minimal time to redirect application servers traffic

Cons

• High hardware setup and administration cost for secondary DB

• Overhead to create and maintain Last Modified Date column

• High overhead to sync up all modified records since secondary DB replication
Alternative 3: In-line Conversion WITHOUT Status column

1. Scan to identify all records with 8 bit Cp1252(0x80 to 0xFF) textual data to be converted
2. Use a script to convert all identified records to UTF-8
3. Repeat Step 1 & 2 to convert newly generated records with Cp1252 data until there is only a small number of Cp1252 records
4. During DB outage window, scan for remaining records with 8 bit Cp1252 textual data and convert them to UTF-8
5. ALTER DATABASE CHARSET from WE8ISO8859P1 to UTF8
Alternative 3: In-line Conversion WITHOUT Status flag

Pros

• Creation of UTF-8 Status column in each table is not required

• Ability to pause Data Conversion if necessary

• Ability to rollback ALTER DB if necessary

• Does not require additional hardware for secondary DB

Cons

• Implement charset detection logic into site code base could increase code complexity and performance overhead

• Charset detection to process each record is less reliable than explicit UTF-8 Status flag

• Multiple iteration of data conversion process

• Potential longer DB outage to convert remaining Cp1252 records during the final step of data conversion