Denormalized Data
Lesson 8: Denormalized Data

Introduction

Lesson aim
This lesson shows you the most common types of denormalization with examples.

Overview

- Denormalization
- Benefits
- Types of denormalization

<table>
<thead>
<tr>
<th>Topic</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why and When to Denormalize</td>
<td>4</td>
</tr>
<tr>
<td>Storing Derivable Values</td>
<td>6</td>
</tr>
<tr>
<td>Pre-Joining Tables</td>
<td>8</td>
</tr>
<tr>
<td>Hard-Coded Values</td>
<td>10</td>
</tr>
<tr>
<td>Keeping Details With Master</td>
<td>12</td>
</tr>
<tr>
<td>Repeating Single Detail with Master</td>
<td>14</td>
</tr>
<tr>
<td>Short-Circuit Keys</td>
<td>16</td>
</tr>
<tr>
<td>End Date Columns</td>
<td>18</td>
</tr>
<tr>
<td>Current Indicator Column</td>
<td>20</td>
</tr>
<tr>
<td>Hierarchy Level Indicator</td>
<td>22</td>
</tr>
<tr>
<td>Denormalization Summary</td>
<td>24</td>
</tr>
<tr>
<td>Practice 8—1: Name that Denormalization</td>
<td>25</td>
</tr>
<tr>
<td>Practice 8—3: Denormalize Price Lists</td>
<td>29</td>
</tr>
<tr>
<td>Practice 8—4: Global Naming</td>
<td>30</td>
</tr>
</tbody>
</table>

Schedule
See Page 31

About the slide
See Page 31
Objectives

At the end of this lesson, you should be able to do the following:

- Define denormalization and explain its benefits
- Differentiate and describe the different circumstances where denormalization is appropriate
Why and When to Denormalize

Definition of Denormalization
Denormalization aids the process of systematically adding redundancy to the database to improve performance after other possibilities, such as indexing, have failed. You will read more on indexing in the lesson on Design Considerations. Denormalization can improve certain types of data access dramatically, but there is no success guaranteed and there is always a cost. The data model becomes less robust, and it will always slow DML down. It complicates processing and introduces the possibility of data integrity problems. It always requires additional programming to maintain the denormalized data.

Denormalization Overview

Denormalization

- Starts with a “normalized” model
- Adds “redundancy” to the design
- Reduces the “integrity” of the design
- Application code added to compensate

Hints for Denormalizing

- Always create a conceptual data model that is completely normalized.
- Consider denormalization as the last option to boost performance.
- Never presume denormalization will be required.
- To meet performance objectives, denormalization should be done during the database design.
- Once performance objectives have been met, do not implement any further denormalization.
- Fully document all denormalization, stating what was done to the tables, what application code was added to compensate for the denormalization, and the reasons for and against doing it.
Denormalization Techniques and Issues

In the next pages you see a number of denormalization techniques that are used regularly. For every type of denormalization you see an indication of when it is appropriate to use it and what the advantages and disadvantages are.

The following topics are covered:

- Storing Derivable Values
- Pre-joining Tables
- Hard-Coded Values
- Keeping Details with Master
- Repeating Single Detail with Master
- Short-Circuit Keys

and the most common specific examples:

- Derivable End Date Column
- Derivable Current Indicator column
- Hierarchy Level Indicator
Storing Derivable Values

When a calculation is frequently executed during queries, it can be worthwhile storing the results of the calculation. If the calculation involves detail records, then store the derived calculation in the master table. Make sure to write application code to re-calculate the value, each time that DML is executed against the detail records.

In all situations of storing derivable values, make sure that the denormalized values cannot be directly updated. They should always be recalculated by the system.

Appropriate:
- When the source values are in multiple records or tables
- When derivable values are frequently needed and when the source values are not
- When the source values are infrequently changed

Advantages:
- Source values do not need to be looked up every time the derivable value is required
- The calculation does not need to be performed during a query or report

Disadvantages:
- DML against the source data will require recalculation or adjustment of the derivable data
- Data duplication introduces the possibility of data inconsistencies

About the slide
See Page 32
When a message is delivered to a recipient, the user only receives a pointer to that message, which is recorded in RECEIVED_MESSAGES. The reason for this, of course, is to prevent the mail system from storing a hundred copies of the same message when one message is sent to a hundred recipients.

Then, when someone deletes a message from their account, only the entry in the RECEIVED_MESSAGES table is removed. Only after all RECEIVED_MESSAGE entries, for a specific message, have been deleted, should the actual message be deleted too.

We could consider adding a denormalized column to the MESSAGES table to keep track of the total number of RECEIVED_MESSAGES that are still kept for a particular message. Then each time users delete a row in RECEIVED_MESSAGES, in other words, they delete a pointer to the message, the Number_of_times_received column can be decremented. When the value of the denormalized column equals zero, then we know the message can also be deleted from the MESSAGES table.
Pre-Joining Tables

You can pre-join tables by including a nonkey column in a table, when the actual value of the primary key, and consequentially the foreign key, has no business meaning. By including a nonkey column that has business meaning, you can avoid joining tables, thus speeding up specific queries.

You must include application code that updates the denormalized column, each time the “master” column value changes in the referenced record.

Appropriate:
- When frequent queries against many tables are required
- When slightly stale data is acceptable

Advantages
- Time-consuming joins can be avoided
- Updates may be postponed when stale data is acceptable

Disadvantages
- Extra DML needed to update original nondenormalized column
- Extra column and possibly larger indices require more working space and disk space
Example
Suppose users often need to query RECEIVED_MESSAGES, using the name of the folder where the received message is filed. In this case it saves time when the name of the folder is available in the RECEIVED_MESSAGES table.

Now, if a user needs to find all messages in a particular folder, only a query on RECEIVED_MESSAGES is needed.

Clearly, the disadvantage is extra storage space for the extra column in a, potentially, very large table.
Hard-Coded Values

If a reference table contains records that remain constant, then you can consider hard-coding those values into the application code. This will mean that you will not need to join tables to retrieve the list of reference values. This is a special type of denormalization, when values are kept outside a table in the database. In the example, you should consider creating a check constraint to the B table in the database that will validate values against the allowable reference values. Note that a check constraint, though it resides in the database, is still a form of hardcoding.

Whenever a new value of A is needed the constraint must be rewritten.

Appropriate
- When the set of allowable values can reasonably be considered to be static during the life cycle of the system
- When the set of possible values is small, say, less than 30

Advantages
- Avoids implementing a look-up table
- Avoids joins to a look-up table

Disadvantages
- Changing look-up values requires recoding and retesting
Example

ElectronicMail would like to know some background information about their users, such as the type of business they work in. Therefore EM have created a table to store all the valid BUSINESS_TYPES they want to distinguish. The values in this table are set up front and not likely to change.

This is a candidate for hard-coding the allowable values. You could consider placing a check constraint on the column in the database. In addition to that, or instead of that, you could build the check into the field validation for the screen application where users can sign in to the EM service.
Keeping Details With Master

In a situation where the number of detail records per master is a fixed value (or has a fixed maximum) and where usually all detail records are queried with the master, you may consider adding the detail columns to the master table. This denormalization works best when the number of records in the detail table are small. This way you will reduce the number of joins during queries. An example is a planning system where there is one record per person per day. This could be replaced by one record per person per month, the table containing a column for each day of the month.

Appropriate
- When the number of detail records for all masters is fixed and static
- When the number of detail records multiplied by the number of columns of the detail is small, say less than 30

Advantages
- No joins are required
- Saves space, as keys are not propagated

Disadvantages
- Increases complexity of data manipulation language (DML) and SELECTs across detail values
- Checks for Amount column must be repeated for Amount1, Amount2 and so on
- Table name A might no longer match the actual content of the table
Example

Suppose each e-mail user is assigned two quotas—one for messages and one for files. The amount of each quota is different, so both have to be tracked individually. The quota does not change very frequently. To be relationally pure, we would create a two-record STORAGE_TYPES table and a STORAGE_QUOTAS table with records for each user, one for each quota type. Instead, we can create the following denormalized columns in the USER table:

- Message_Quota_Allocated
- Message_Quota_Available
- File_Quota_Allocated
- File_Quota_Available

Note that the name of table USERS does not really match the data in the denormalized table.
Lesson 8: Denormalized Data

Repeating Single Detail with Master

Often when the storage of historical data is necessary, many queries require only the most current record. You can add a new foreign key column to store this single detail with its master. Make sure you add code to change the denormalized column any time a new record is added to the history table.

Appropriate
• When detail records per master have a property such that one record can be considered “current” and others “historical”
• When queries frequently need this specific single detail, and only occasionally need the other details
• When the Master often has only one single detail record

Advantages
• No join is required for queries that only need the specific single detail

Disadvantages
• Detail value must be repeated, with the possibility of data inconsistencies

Additional code must be written to maintain the duplicated single detail value at the master record.
Example

Any time a message is sent, it can be sent with attachments included. Messages can have more than one attachment. Suppose in the majority of the messages that there is no or only one attachment. To avoid a table join, you could store the attachment name in the MESSAGES table. For those messages containing more than one attachment, only the first attachment would be taken. The remaining attachments would be in the ATTACHMENTS table.
Short-Circuit Keys

For database designs that contain three (or more) levels of master detail, and there is a need to query the lowest and highest level records only, consider creating short-circuit keys. These new foreign key definitions directly link the lowest level detail records to higher level grandparent records. The result can produce fewer table joins when queries execute.

Appropriate
- When queries frequently require values from a grandparent and grandchild, but not from the parent

Advantages
- Queries join fewer tables together

Disadvantages
- Extra foreign keys are required
- Extra code is required to make sure that the value of the denormalized column A_id is consistent with the value you would find after a join with table B.
Example

Suppose frequent queries are submitted that require data from the RECEIVED_MESSAGES table and the USERS table, but not from the FOLDERS table. To avoid having to join USERS and FOLDERS, the primary or a unique key of the USERS table can be migrated to the RECEIVED_MESSAGES table, to provide information about USERS and RECEIVED_MESSAGES with one less, or no, table join.
End Date Columns

The most common denormalization decision is to store the end date for periods that are *consecutive*; then the end date for a period can be derived from the start date of the previous period.

If you do this, to find a detail record for a particular date you avoid the need to use a complex subquery.

**Appropriate**
- When queries are needed from tables with long lists or records that are historical and you are interested in the most current record

**Advantages**
- Can use the between operator for date selection queries instead of potentially time-consuming synchronized subquery

**Disadvantages**
- Extra code needed to populate the end date column with the value found in the previous start date record
Example

When a business wishes to track the price history of a product, they may use a PRICES table that contains columns for the price and its start date and a foreign key to the PRODUCTS table. To avoid using a subquery when looking for the price on a specific date, you could consider adding an end date column. You should then write some application code to update the end date each time a new price is inserted.

Compare:

```sql
...WHERE pdt_id = ...
AND start_date = ( SELECT max(start_date)
FROM prices
WHERE start_date <= sysdate
AND pdt_id = ...
)
```

and

```sql
...WHERE pdt_id = ...
AND sysdate between start_date and nvl(end_date, sysdate)
```

Note that the first table structure presupposes that products always have a price since the first price start date of that product. This may very well be desirable but not always the case in many business situations.

Note also that you would need code to make sure periods do not overlap.
Lesson 8: Denormalized Data

Current Indicator Column

This type of denormalization can be used in similar situations to the end date column technique. It can even be used in addition to an end date. It is a very common type of denormalization.

Suppose most of the queries are to find the most current detail record. With this type of requirement, you could consider adding a new column to the details table to represent the currently active record.

You would need to add code to update that column each time you insert a new record.

Appropriate
- When the situation requires retrieving the most current record from a long list

Advantages
- Less complicated queries or subqueries

Disadvantages
- Extra column and application code to maintain it
- The concept of “current” makes it impossible to make data adjustments ahead of time
Example

In the first table structure, when the current price of a product is needed, you need to query the PRICES table using:

```sql
...WHERE pdt_id = ... AND start_date = ( SELECT max(start_date)
FROM prices
WHERE start_date <= sysdate
AND pdt_id = ... )
```

The query in the second situation would simply be:

```sql
...WHERE pdt_id = ... AND current_indicator = ‘Y’
```
Lesson 8: Denormalized Data

**Hierarchy Level Indicator**

Suppose there is a business limit to the number of levels a particular hierarchy may contain. Or suppose in many situations you need to know records that have the same level in a hierarchy. In both these situations, you will need to use a connect-by clause to traverse the hierarchy. This type of clause can be costly on performance. You could add a column to represent the level of a record in the hierarchy, and then just use that value instead of the connect-by clause in SQL.

**Appropriate**

- When there are limits to the number of levels within a hierarchy, and you do not want to use a connect-by search to see if the limit has been reached
- When you want to find records located at the same level in the hierarchy
- When the level value is often used for particular business reasons

**Advantages**

- No need to use the connect-by clause in query code

**Disadvantages**

- Each time a foreign key is updated, the level indicator needs to be recalculated, and you may need to cascade the changes

---

**About the slide**

See Page 33
Example
Imagine that because of storage limitations, a limit has been placed on the number of nested folders. Each time a user wants to create a new instance of a folder within an existing folder instance, code must decide if that limit has been reached. This can be a slow process.

If you add a column to indicate at what nested level a FOLDER is, then when you create a new folder in it, you can decide immediately if this is allowed. If it is, the level of the new folder is simply one more than the level of the folder it resides in.
Lesson 8: Denormalized Data

Denormalization Summary

Denormalization is a structured process and should not be done lightly. Every denormalization step will require additional application code. Be confident you do want to introduce this redundant data.

**Denormalization Summary**

Denormalization Techniques

- Storing Derivable Information
  - End Date Column
  - Current Indicator
  - Hierarchy Level Indicator
- Pre-Joining Tables
- Hard-Coded Values
- Keeping Detail with Master
- Repeating Single Detail with Master
- Short-Circuit Keys