Information Sharing With the Oracle Database

Deiter Gawlick
Oracle Corporation
500 Oracle Parkway
Redwood City CA 94065
(650) 506-8706
deiter.gawlick@oracle.com

Shailendra Mishra
Oracle Corporation
500 Oracle Parkway
Redwood City 94065
(650) 506-9123
shailendra.mishra@oracle.com

ABSTRACT

Database systems have been designed to manage business critical information and make this information accessible on request to connected clients. There is, however, an ever-increasing need to share relevant information actively with disconnected clients and/or external systems, e.g., to propagate and/or automatically react to relevant information as soon as it becomes available.

Leveraging the existing database infrastructure, Oracle created a solution to this problem. The solution is state of the art in terms of functionality and operational characteristics. This Oracle technology is widely used and supports many highly visible applications.

Categories and Subject Descriptors

H. INFORMATION SYSTEMS

H.4. INFORMATIONS SYSTEMS APPLICATIONS

H.4.3 Communications Applications

General Terms

Design.

Keywords

Database, Type system, Publish/Subscribe, JMS, Reliable Messaging, Post Dating, Rules, Capture, Apply, Retention, Auditing, Expression evaluation.

1. INTRODUCTION

The ability to share information easily and in a timely fashion is a crucial requirement for any business environment. Consequently, *Information Sharing* has been supported by many mechanisms, e.g., discussions, mail, books, periodicals, and last but not least, computer technology.

Many computer-based technologies have evolved to serve Information Sharing. The ability to always be in contact, especially with the evolution of wireless communication devices, brings increased urgency to this problem of active notification of relevant information. The requirement is always the same*: 'I want to know what is going on - based on my perception of the world - and I want to know it ASAP and wherever I am.'* Unfortunately, most Information Sharing solutions are still based on significant application logic, resulting in what is most likely the most expensive means to develop, deploy, operate, and maintain these services. Additionally, these services often lack required functionality, such as support for ad-hoc requests and customization, as well as timely and flexible delivery of information.

Since databases manage a significant part of the business critical information, it is only natural to provide the required support from within databases. One obvious benefit is the ability to leverage the rich functionality and the exceptional operational characteristics of the underlying technology.

Oracle started providing a solution by delivering the first queuing system that is fully integrated in the database [ORAQ]. This queuing system, called Oracle AQ (Oracle Advanced Queuing) has been available since Oracle8. Oracle AQ includes a novel idea to communicate with clients, and a novel approach to publish/subscribe. Details of this approach are described in section 2. Section 2 also describes the required database enhancements and the operational benefits of this approach.

Oracle9i extends the existing approach in many ways. Most importantly, clients can determine what they are interested in, thus, subscribers do not depend on publishers to publish the right information. This in effect adds a new model complementing publish/subscribe. This model may be called a *subscribe/publish* model. These services are provided as Oracle Streams [ORSTR]. Section 4 describes the concepts and major implementation challenges, as well as the many benefits of Oracle Streams. Oracle AQ is now a part of Oracle Streams.

Oracle Streams offers the expression evaluation as an independent service [ORRE]. Users can specify subscriptions and data structures. A specific instance of a data structure can be evaluated against a set of subscriptions. This support allows applications to make a publication dependent on the interest of subscribers, and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Conference '03, Month 6-7, 2003, San Diego, CA.
Copyright 2000 ACM 1-58113-000-0/00/0000…$5.00.

to tailor publications exactly to the needs of the subscribers. Details of this support are described in section 3.

More recently, Oracle has discussed expressions as data in relational databases [YSG]. Such functionality provides additional flexibility for Information Sharing as shown in section 5.

2. Oracle AQ

It is assumed that readers are familiar with technology such as JMS [JMS]. Oracle AQ is fully JMS compliant. However, Oracle AQ extends the JMS model – leveraging the JMS extensibility framework. The main objectives for the functional extensions are:

- Provide an improved communication model
- Provide an improved publish/subscribe model

The management and delivery of messages is the core functionality of messaging systems. Messages are stored in containers called queues. Queues are typically mapped to files. Oracle AQ is unique since it maps queues to database tables using a special tables called queue table. Queue tables are like regular database tables, with a specific data structure. This data structure includes among other things contains columns for message header and user payload. The payload can be any structure that is supported by the database type system. Additionally, there are interfaces (ENQUEUE, DEQUEUE) to create and consume messages.

All messaging systems allow publishers to add (ENQUEUE) messages to queues and consumers to consume (DEQUEUE) messages, assuming they have the requisite privelege to do so. Oracle AQ also supports this model too. However, there is an additional construct supported by Oracle AQ called the multi-consumer queue. In this, a consumer is an entity that receives messages through any communication channel. A consumer, called a recipient, is described through a channel type and an address for the channel, e.g., email:joe@dot.com, tel:+1-650-506-0000, WSDL:port@oracle.com, AQ:orders@oracle.com, or broadcast:stockticks@NYSE. A message destined for a recipient can then be sent to them through the specified channels. This mechanism allows a publisher to use any available communication channel to reach a desired recipient, recipient groups, or any mix there-of. Users can add propagation support for any communication channel that is not supported. A multi-consumer queue allows consumers to consume only those messages that are directed to them.

Multi-consumer queues provide support for *directed message distribution*.

Recipients can be specified independent of the specific communication channel, assuming there is a recipient directory. In this case, only a list of recipient names needs to be specified. Oracle AQ will translate the names into the addressing specification and direct the information to the intended recipients using the specified communication channel and address.

Alternatively, publishers may not wish to determine who gets a message and may leave the choice to the recipients, thus, the recipient become subscribers. Oracle AQ allows consumers to create subscriptions. A subscription consists of a reference to a queue, a condition specifying which messages are of interest, and a list of recipients.

Since Oracle AQ stores messages as rows in tables, SQL WHERE clauses can be used to specify the conditions for selecting the desired messages. These SQL WHERE clauses can reference any part of a message, Subscriptions without a SQL WHERE clause represent a subject-based subscription. A subscription with a WHERE clause is a content based subscription.

The idea behind this model is quite simple. Subscriptions are queries on future data. Consequently, subscribers can leverage their SQL knowledge to specify subscriptions.

Messages can be structured using SQL, XML, or the extensibility support (e.g., Text, Spatial) using the full language support for each of these structures.

The database security mechanism is used to determine the base level per subscriber visiblity of a message. Optionaly additional protection for messages can be achieved using the fine grained security model.

This Oracle AQ model represents a significant generalization of the JMS specification and is implemented in accordance to the JMS extensibility framework [JMS].

The database constructs had to be enhanced to support the messaging system. Here is a list of some of the major enhancements:

- SKIP LOCKED – Messages are typically ordered by some criteria, e.g., the time of creation. The SQL WHERE clause, which is used to process a DEQUEUE request, uses ORDERED BY to retrieve messages in the desired order. Unfortunately, ORDERED BY allows only one message consumer at a time. SKIP LOCKED has been added to direct the database to serve several consumers concurrently and to skip those messages that are already in the process of being consumed by other consumers. The first available message in the specified order will be selected.

- WAIT FOR – Consumers can request a new message, regardless of whether one is available for consumption or not. The messaging system has to block the consumer process until a message becomes available. Databases do not have this concept. If data do not exist, databases immediately return with a 'NOT FOUND' indication. This would force consumers to pull for messages, wasting resource. If a DEQUEUE does not find any messages, the consumer process will be blocked and will be POSTED as soon as a new message is added (and committed) to the queue. Consumers can specify how long they are willing to wait; the value zero prevents any wait. There are additional optimizations, such as waiting for a new message arriving in any of several queues and/or registering to the arrival of new messages. A registration leads to a notification and prevents blocking.

- Rules Processing – SQL engines are built to find data in response to a query. Subscription engines are built to find queries (expressions, WHERE clauses) in response to data. Since the objective is to make subscriptions react exactly like queries – on future data – the SQL engine had to be used as much as possible This is exactly what has been done. Once a subscription is specified, the SQL compilation process is executed to check for syntax errors and access violations. When data are presented for subscription evaluation, Oracle

AQ first searches a subscription index. Any subscription that is potentially valid will be evaluated against the presented data. This is done with a minor modification of the existing technology, the engine loops through the selected subscription. In contrast, when processing queries the engine loops through the data, which are selected after the (data) index lookup. This ensures that any operation that is supported by queries is also supported by subscriptions, and that subscriptions act indeed as queries on future data.

- Index maintenance – Indices are used to control efficient access to queues. Indices in message system have to deal with an unusually high insertion and deletion rate. As a consequence indices become very unbalanced. The database server deals with this issue by periodically rebalancing an index whenever it is required. Actually, queue indices are represented as IOT's (Index Organized Tables) [SRI]. IOT's allow user data to be physically co-located with indices, which leads to improved performance.

A side effect of using a database messaging system is improved functionality and improved operational characteristics. Here are some examples of these improvements.

- Type system – Messages can be unstructured, semi-structured, or structured. The support includes SQL92, SQL99, and XML as well as text, spatial data, or any user specified data structures. BLOBs and CLOBs provide support for very large messages.
- Data access – The (SQL) query support can be used to find messages. There is no limitation. This support applies to any message structure.
- Subscription specification – Subscriptions are treated as queries on future data. Consequently, users have a rich (standard) language to subscribe to messages. Consequently, subscriptions behave as much as possible like queries.
- Postdating – Messages can be published for consumption in the future. Together with the standard expiration support, users can specify a window of processing. Messages will only be visible with the standard queuing interface (DEQUEUE) once the time of processing has been reached.
- Retention - Oracle AQ allows users to retain messages after they have been consumed. While these messages are no longer visible with the queuing interface (DEQUEUE), they are still visible using the standard data access methods.
- Auditing and tracking – Using standard data access, users can audit and track messages. This applies to messages in all states, post dated, ready to be processed, and consumed.
- Reliability – Since messages are rows in tables they inherit the reliability of the database. This includes restart and recovery as well as hot swaps, fault tolerance, and disaster protection. This is due to the RAC (Real Application Cluster) technology [ORRAC].
- Scalability – Since databases are able to handle large numbers of tables with very large numbers of potentially very large rows, the size and the number of messages as well as the number of queues can be very large. Due to Oracle's RAC technology Oracle AQ can run on several processing nodes or on blades and therefore supports a very large user community.
- Performance – Databases are designed for high performance. Messages are often published and consumed in the context of transactions. Since Oracle AQ is part of the Oracle database, there is no need for a distributed two-phase commit process. In business environments, this typically results in a performance improvement by a factor two to three.
- Security – All the security features of the database apply to Oracle AQ, including fine grain security, which includes VPN (Virtual Private Database) support. This allows users to control access to messages down to the level of individual messages.
- TCO (Total Cost of Ownership) – The integration of messaging in the Oracle database, significantly reduces the cost of ownership. This is due to reduced effort in programming (more functionality and shared client with database), installation (there is nothing to install and no compatibility to test), and operation (there is no additional operational environment).

1. Expression Evaluation

Expression evaluation is typically an internal service of a publish/subscribe or a message distribution system. There are, however, many applications where users would like to find if there is interest in specific information, e.g., sophisticated applications are able to publish highly customized information. Only a very small subset of the possible publications is of interest for at least one subscriber. So, the best strategy would be to reverse the sequence and create an event (a publication) only if there is a subscriber.

A stand-alone service to do rules evaluations [ORRE] provides publishers with increased functionality and operational characteristics. In the rules evaluation service, a rule set is defined as a collection of rules, and represents a unit of evaluation. A rule set may have an associated evaluation context, which can be used by any element of the rule set. Other elements of the rules framework are described below:

- A rule consists of a rule condition, an optional evaluation context, and an optional action context.
- A rule condition is a valid Boolean SQL expression. It has the ability to refer to data stored persistently or non-persistently.
- The evaluation context consists of any information necessary to interpret a rule. For example, if the rule refers to the column names of a table, then the evaluation context provides a list of such table column table names. A rule thus may have an evaluation context or may inherit an evaluation context from a rule set. (A rule set is a collection of rules.)
- The action context of a rule consists of a list of name-value pairs that is interpreted by the application using the rule. It reflects the fact that rules often are applied in an event-based framework, where events lead to actions based on rules that are satisfied.

An independent rules evaluation service, a Rules Engine, provides publishers with much increased functionality and operational characteristics. By using a rules engine, publishers can use any information to evaluate subscriptions, not just the publications. Publications need only be created if there is an interested subscriber. Additionally, using the action context, publications can be tailored to the need of the subscribers.

2. Oracle Streams

Oracle Streams [ORSTR] is designed to satisfy the requirements of messaging and database replication. This paper will focus on three features of Oracle Streams that resulted from this design point and created significant benefits to message users:

- Automatic publications – the capture process.
- Automatic consumption – the apply process.
- Type any queues.

Information Sharing needs to deal with the following three major tasks, as defined by the ECA (Event Condition Action) model [DBC]:

- The publication of information – these are the events
- The evaluation of conditions to activate the proper action on the published information
- The execution of the selected actions. These actions include execution of procedures, transformation of data, and/or distribution of information.

Existing support focused on solving the last two tasks and did not provide any help for the creation of the publication– the event in the language of the ECA model. Consequently, there is no good model to ensure the publication of the 'right' information in a timely manner.

Fortunately, for some very important applications, it is possible to automatically publish the right information in response to subscriptions. Changes in databases are one of these cases.

Oracle Streams uses the WHERE clauses in the subscription as queries against records of the system journal. The journal records that are of interest to at least one subscriber are published according to directives in the subscription. Once the publication is completed, any specified action (procedures, transformation and/or information distribution) is executed.

In the language of the ECA model, the condition is used as the publication criteria and not as the selection criteria against a publication.

The journal represents an interesting source for publication. The journal is not only able to publish on demand and according to the subscriber requirements, but is also able to execute and repeat the publication on demand. This leads to a *pull* publication; publications are typically push oriented.

Since the action is known in the context of data base replication, Oracle Streams also provides an automatic consumption process, called the apply process. An apply process consumes published information to complete the replication.

While the support for an automatic publication and consumption is essential for the replication support, it can be used in other environments. Users can design capture and apply processes as part of their application logic, as is done in the Oracle e-Business Suite [OREBIS].

A database instance often has a large number of tables, each of which typically has its own type definition. Using typed queues would require at least as many queues as there are tables to replicate. Oracle Streams solves this problem by supporting tables that can store any data types that have been defined for the database instance. Thus, users can specify queue tables of type ANYDATA.

Type ANY queues free users from the need to align queue definitions with type definitions and lead to a potentially significant reduction of queues.

Capture can repeat the publication. Apply processes used for replication can determine whether a publication has been applied already or not. This leads to an important optimization. Oracle Streams supports the model of *buffered* queues. Buffered queues are used to act as buffers between the capture processes and propagation and between propagation and the apply processes. Buffered queues are transient queues, that is, there are queues without journaling with the exception of writing periodic checkpoints containing information about the status of the apply processes. However, the interaction between capture processes, apply processes, and buffered queues provides exactly once semantics.

After a failure, using the checkpoint information, Oracle Streams determines which information has to be re-published to the apply processes. This information, if no any longer available in buffered queues, will be re-published on request by the capture processes. The apply processes will ignore duplicates arising from processing published information from the selected checkpoint and the failure.

Details of the implementation are complex and are beyond the scope of this paper.

Oracle Streams provides an infrastructure to support many information-sharing environments using a small number of building blocks:

- Automated (implicit) or explicit capture processes
- Staging areas (queues) for published information. These staging areas can be transient or persistent with retention, auditing, tracking support.
- Automated (implicit) or explicit apply processes
- The ability to propagate information between staging areas
- The ability to receive/deliver information from/to communication channels

Using these building blocks, the following scenarios can be supported:

- Basic messaging – this scenario is handled with explicit capture and apply processes using standard or buffered queues of type ANYDATA. Standard queues would be used without retention.
- Directed message distribution – this scenario is handled with explicit capture and apply processes using multi-consumer queues that are typed or of type ANYDATA.

- Messaging with auditing and tracking – this scenario is handled with explicit capture and apply processes using queues that are typed and support retention. This would be the normal mode of operation between autonomous business entities, especially for any EAI environments.
- Publish/subscribe (messaging context) - this scenario is handled with explicit capture and apply processes using typed multi-consumer queues. Publishers do not specify the recipients. The recipient list is determined through subscriptions. The support includes subject and content based subscription, that is, subscription that are specified without or with WHERE clause respectively. When using multi-consumer queues, publishers can choose to select the recipients or not on a message-by-message basis. If they specify recipients for a message, subscriptions will be ignored. If they do not specify recipients subscription will be used to determine them.
- Replication (homogeneous) - this scenario is handled with implicit capture and apply processes using buffered queues. Subscriptions are used to determine which messages are published. Messages will be re-published as needed. Buffered queues are used for high performance.
- Replication (heterogeneous) - this scenario is handled with implicit capture and apply processes using buffered queues. Subscriptions are used to determine which messages are published. Messages will be re-published as needed. Buffered queues are used for high performance. Gateway technology is used for the apply processes.
- Replication (generalized) - this scenario is handled with implicit capture and apply processes using buffered queues. Rule-based transformations during capture, propagation, and/or apply processing can support different table names, different column names, and different column data types.
- Low latency ETL for Data Warehouse – this scenario is handled like the generalized replication case. It leverages the high flexibility, and functionality of rules based selections and transformation. ETL (Extraction, Transformation, Loading) based on Oracle Streams technology provides up-to-date information. Furthermore the extracted information includes the evolution of the data, not just a specific state. Last not least, information of the past can be extracted.
- Rolling application upgrades – this scenario is handled like the generalized replication case. If a new version of an application needs different data structures, the flexibility and functionality of rules based transformation can be leveraged to transfer the data from the old model into a new model. Tables representing the data in the new model are created and kept up-to-date. Once this process is completed, the new application version can be activated.

  The transformation of data may take days and therefore require a service interrupt of unacceptable duration. This technology reduces the outage to an acceptable level.

  It should be noted that sometimes not all transformations could be handled as online transformation. In these cases some 'cleanup' has to be done while the application is not active. Experience has shown that there are typically only small of data that cannot be handled online and that the interruption of the application to transform this data can be tolerated.
- Disaster protection – this scenario is handled with implicit capture and apply processes using buffered queues. Subscriptions are used to determine which information has to be delivered from the active site to the remote stand-by site.

The above scenarios show how a single infrastructure can be used for a range scenarios covering and significantly extending existing messaging, publish/subscribe, and replication technology.

3. Expressions as Data

Conditional expression can be used to describe interest in information. One could consider expressions in isolation, as publish/subscribe systems generally do, or as part of a more general description of message consumers.

Let us assume there is a directory in the form of a table describing message consumers. This directory consists of multiple columns, such as user name, user identification, address of user, communication channel with addresses, their company affiliation, position within the company, and their interest and/or responsibilities. Let us assume that for greater flexibility and expressiveness expressions are used to specify the interest and responsibilities.

One could issue a query to find rows that satisfy certain criteria, for example, find everyone from a company who is located within a certain radius from a specific point. The result set could be used to define the consumers of a message. In this case the publisher determines the recipients with a SELECT on the table.

Expression Filtering as described in [YSG] allows users to find those rows for which the expression matches a given data element. More formally:

```
SELECT * FROM consumers WHERE
      EVALUATE (consumer.Interest) …,
               <data item>) = 1
```

This statement returns those rows for which the expressions match the values of the data item. If the data item represents a message and the expression represents a subscription, we would find all the subscribers of a message. The publisher uses the criteria of consumers to specify the recipients.

Obviously, the data item does not need to be a message, so one could ask the question: 'Is there interest in some information,' and publish the information only if this is the case. This represents a simple demand analysis for information.

Expression Filtering allows publishers to specify queries that reference any combinations of columns including expressions. This allows publishers to use their own criteria as well as the criteria of the subscribers; functionality called mutual filtering. The SQL support allows adding more sophisticated distribution lists, such as, the first ten recipients ordered by distance from some location.

Expression filtering can be applied to XML messages by capturing the interest as XPath expressions..

Obviously, there are many application of this technology outside of Information Sharing. A good example would be demand analysis. A seller could check demand for set a of items by a set of consumers by specifying a query against a customer database that contains consumer interest as a part of consumer data.

4. Conclusions

Messaging and database technology have evolved independently. There have been early attempt to use databases to store messages without adapting the database technology to the messaging environments [NEON]. These attempts resulted in severe performance and functional limitations.

Oracle's approach finally brought the breakthrough. The approach presented in this paper achieved the following benefits

- Significantly extended the information sharing functionality
- Significantly improved the operational characteristics
- Significantly reduced TCO (Total Cost of Ownership) for users
- Significantly reduced development time and effort for Oracle

The Oracle effort confirmed the perception of many researchers that messaging and database technology should be integrated [BHM], [GR]. It also adds a new perspective to the work related to active databases as described in [WC].

5. Acknowledgement

The technology, described in this paper, has been deployed widely among Oracle's customers. It is the backbone of many mission critical and highly visible applications. It has become an integral part of all of Oracle's software including the E-Business Suite and the Oracle Collaboration Server.

The development and implementation of the technology is the result of many years of many dedicated Oracle employees. Neerja Bhatt, Alan Downing, and Jim Stamos are key contributors.

6. References

[BHM] – Bernstein, P., Hsu, M. and Mann, B, "Implementing Recoverable Requests Using Queues.' 1990 ACM SIGMOD Conference, Atlantic City, May 1990

[DBC] - Dayal, U., Buchmann, A., and Chakravarthy, S., "The HIPAC Project" in "Active Database Systems," *Morgan-Kaufmann, San Mateo, California*
1995.
GR] – Gray, J., Reuter, A., "Transaction Processing,." *Morgan-Kaufmann, San Mateo, California*, 1993.
[JMS] - http://java.sun.com/ (Search 'JMS')

[NEON] - http://www.sybase.com/products (search for' New Era of Networks Integration Package')

[ORRAC] - http://otn.oracle.com/ (Search for 'RAC')

[ORAQ] - http://otn.oracle.com/ (Search for 'Oracle AQ')

[OREBIS] - http://otn.oracle.com (Search for 'Workflow, Business Events')

[ORRE] - http://otn.oracle.com/ (Search for 'Rules Engine')

[ORSTR] - http://otn.oracle.com/ Search for 'Oracle Streams')

[SRI] Srinivasan, J., et.al., "Oracle8*i* Index-Organized Table and its Applications to New Domains," *Proceedings of the 26th Int. Conf. on Very Large Data Bases*, pp. 285-296, Sept. 2000.

[WC] Widom, J. and Ceri, S. "Active Database Systems". *Morgan-Kaufmann, San Mateo, California*, *1995*.
[YSG] – Yalamanchi, A., Srinivasan, J., and Gawlick, D., "Managing Expressions as Data in Relational Database Systems," CIDR Conference, Asilomar, 2003