Oracle CDC调研报告

林子雨

北京大学计算机系数据库实验室

2006年11月10日

The Redo Log and a Capture Process

- *Redo log*
  - Every Oracle database has a set of two or more *redo log* files.
  - The *redo log* files for a database are collectively known as the database *redo log*.
  - The primary function of the *redo log* is to record all changes made to the database.
  - *Redo logs* are used to guarantee recoverability in the event of human error or media failure.
- *Capture process*
  - A **capture process** is an optional Oracle background process that scans the database redo log to capture DML and DDL changes made to database objects.
  - When a capture process is configured to capture changes from a redo log, the database where the changes were generated is called the **source database**.

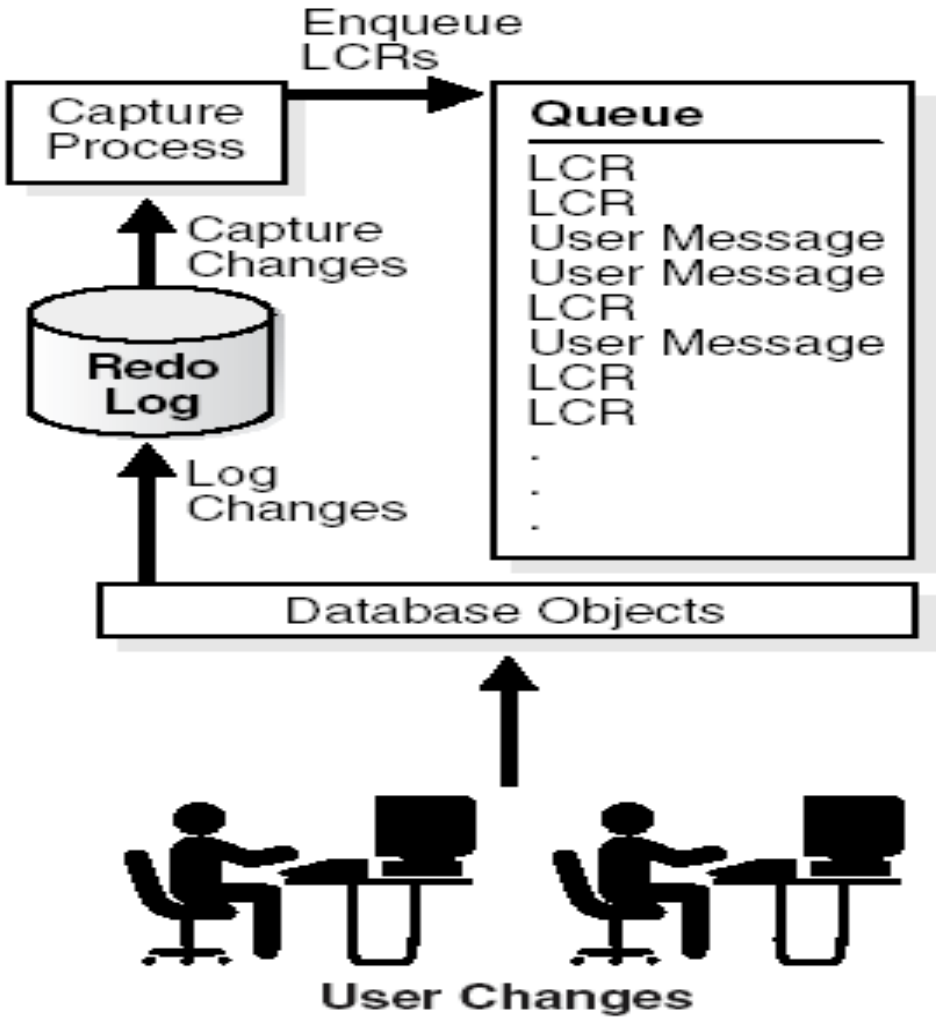
The Redo Log and a Capture Process

- ***local capture process & downstream capture process***
  - A capture process can run on the source database or on a remote database.
  - When a capture process runs on the source database, the capture process is a ***local capture process***.
  - When a capture process runs on a remote database, the capture process is called a ***downstream capture process***, and the remote database is called the **downstream database**.

Logical Change Records (LCRs)

- A **capture process** reformats changes captured from the redo log into LCRs.
- An LCR is a **message** with a specific format that describes a database change.
- A capture process captures two types of LCRs: ***row LCRs*** and ***DDL LCRs***.
- After capturing an LCR, a capture process enqueues a message containing the LCR into a **queue**.
- A capture process is always associated with a single ANYDATA queue, and it enqueues messages into this queue only.
- For improved performance, **captured message**s always are stored in a **buffered queue**, which is System Global Area (SGA) memory associated with an ANYDATA queue. You can create multiple queues and associate a different capture process with each queue.

Logical Change Records (LCRs)



Logical Change Records (LCRs)

Row LCRs

- A ***row LCR*** describes a change to the data in a single row or a change to a single LONG,LONG RAW, or LOB column in a row.
- The change results from a data manipulation language (DML) statement or a piecewise update to a LOB.
- For example, a single DML statement can insert or merge multiple rows into a table, can update multiple rows in a table, or can delete multiple rows from a table.
- Therefore, a single DML statement can produce multiple row LCRs. That is, a capture process creates an LCR for each row that is changed by the DML statement.
- In addition, an update to a LONG, LONG RAW, or LOB column in a single row can result in more than one row LCR.

Logical Change Records (LCRs)

Each row LCR is encapsulated in an object of LCR$_ROW_RECORD type and contains the following attributes:

- ***source_database_name:*** The name of the **source database** where the row change occurred.
- ***command_type:*** The type of DML statement that produced the change, either INSERT, UPDATE, DELETE, LOB ERASE, LOB WRITE, or LOB TRIM.
- ***object_owner:*** The schema name that contains the table with the changed row.
- ***object_name:*** The name of the table that contains the changed row.
- ***tag:*** A raw **tag** that can be used to track the LCR.
- ***transaction_id:*** The identifier of the transaction in which the DML statement was run.

Logical Change Records (LCRs)

Each row LCR is encapsulated in an object of LCR$_ROW_RECORD type and contains the following attributes:

- ***scn:*** The system change number (SCN) at the time when the change record was written to the redo log.

- ***old_values:*** The old column values related to the change. These are the column values for the row before the DML change. If the type of the DML statement is UPDATE or DELETE, then these old values include some or all of the columns in the changed row before the DML statement. If the type of the DML statement is INSERT, then there are no old values.

- ***new_values:*** The new column values related to the change. These are the column values for the row after the DML change. If the type of the DML statement is UPDATE or INSERT, then these new values include some or all of the columns in the changed row after the DML statement. If the type of the DML statement is DELETE, then there are no new values.

Logical Change Records (LCRs)

- A captured ***row LCR*** can also contain transaction control statements.
- These ***row LCRs*** contain directives such as COMMIT and ROLLBACK.
- Such ***row LCRs*** are internal and are used by an **apply process** to maintain transaction consistency between a source database and a **destination database**.

Logical Change Records (LCRs)

- ***DDL LCRs***
  - A ***DDL LCR*** describes a data definition language (DDL) change. A DDL statement changes the structure of the database. For example, a DDL statement can create, alter, or drop a database object.

Logical Change Records (LCRs)

- ***Each DDL LCR contains the following information:***
  - ***source_database_name:*** The name of the **source database** where the DDL change occurred.
  - ***command_type:*** The type of DDL statement that produced the change, for example ALTER TABLE or CREATE INDEX.
  - ***object_owner:*** The schema name of the user who owns the database object on which the DDL statement was run.
  - ***object_name:*** The name of the database object on which the DDL statement was run.
  - ***object_type:*** The type of database object on which the DDL statement was run,for example TABLE or PACKAGE.ddl_text: The text of the DDL statement.
  - ***logon_user:*** The logon user, which is the user whose session executed the DDL statement.

Logical Change Records (LCRs)

- ***Each DDL LCR contains the following information:***
  - ***current_schema:*** The schema that is used if no schema is specified for an object in the DDL text.
  - ***base_table_owner:*** The base table owner. If the DDL statement is dependent on a table, then the base table owner is the owner of the table on which it is dependent.
  - ***base_table_name:*** The base table name. If the DDL statement is dependent on a table, then the base table name is the name of the table on which it is dependent.
  - ***tag:*** A raw **tag** that can be used to track the LCR.
  - ***transaction_id:*** The identifier of the transaction in which the DDL statement was run.
  - ***scn:*** The SCN when the change was written to the redo log.

Capture Process Rules

- *A **capture process** either captures or discards changes based on **rule**s that you define.*

- *Each rule specifies the database objects and types of changes for which the rule evaluates to TRUE. You can place these rules in a **positive rule set** or **negative rule set** for the capture process.*

  - If a rule evaluates to TRUE for a change, and the rule is in the positive rule set for a capture process, then the capture process captures the change.

  - If a rule evaluates to TRUE for a change, and the rule is in the negative rule set for a capture process, then the capture process discards the change. If a capture process has both a positive and a negative rule set, then the negative rule set is always evaluated first.

Capture Process Rules

- ***You can specify capture process rules at the following levels:***
  - A table rule captures or discards either row changes resulting from DML changes or DDL changes to a particular table. Subset rules are table rules that include a subset of the row changes to a particular table.
  - A schema rule captures or discards either row changes resulting from DML changes or DDL changes to the database objects in a particular schema.
  - A global rule captures or discards either all row changes resulting from DML changes or all DDL changes in the database.

Datatypes Captured

- *When capturing the row changes resulting from DML changes made to tables, a **capture process** can capture changes made to columns of the following datatypes:*

VARCHAR2■ NVARCHAR2■ NUMBER■ LONG■ DATE■ BINARY_FLOAT ■ BINARY_DOUBLE■ TIMESTAMP■ TIMESTAMP WITH TIME ZONE ■ TIMESTAMP WITH LOCAL TIME ZONE■ INTERVAL YEAR TO MONTH ■ INTERVAL DAY TO SECOND■ RAW■ LONG RAW■ CHAR■ NCHAR ■ CLOB■ NCLOB■ BLOB■ UROWID

Datatypes Captured

- ***A capture process does not capture the results of DML changes to columns of thefollowing datatypes:***
  - BFILE
  - ROWID
  - user-defined types (including object types, REFs, varrays, nested tables, and Oracle-supplied types).
- ***Also, a capture process cannot capture changes to columns if the columns have been encrypted using transparent data encryption.***

Types of DML Changes Captured

- *When you specify that DML changes made to certain tables should be captured, a capture process captures the following types of DML changes made to these tables:*
  - INSERT
  - UPDATE
  - DELETE
  - MERGE
  - Piecewise updates to LOBs

DDL Changes and Capture Processes

- *A capture process captures the DDL changes that satisfy its **rule sets**, except for the following types of DDL changes:*
  - ALTER DATABASE
  - CREATE CONTROLFILE
  - CREATE DATABASE
  - CREATE PFILE
  - CREATE SPFILE
  - FLASHBACK DATABASE

Instantiation in a Streams Environment

- In a Streams environment that shares a database object within a single database or between multiple databases, a **source database** is the database where changes to the object are generated in the redo log, and a **destination database** is the database where these changes are dequeued by an **apply process**.

- If a **capture process** captures or will capture such changes, and the changes will be applied locally or propagated to other databases and applied at destination databases, then you must **instantiate** these source database objects before these changes can be dequeued and processed by an apply process.

- If a database where changes to the source database objects will be applied is a different database than the source database, then the destination database must have a copy of these database objects.

Instantiation in a Streams Environment

- *In Streams, the following general steps instantiate a database object:*

  - **1.** Prepare the object for instantiation at the source database.

  - **2.** If a copy of the object does not exist at the destination database, then create an object physically at the destination database based on an object at the source database. You can use export/import, transportable table spaces, or RMAN to copy database objects for instantiation. If the database objects already exist at the destination database, then this step is not necessary.

  - **3.** Set the **instantiation SCN** for the database object at the destination database. An instantiation SCN instructs an apply process at the destination database to apply only changes that committed at the source database after the specified SCN.

Local Capture and Downstream Capture

- *You can configure a capture process to run locally on a source database or remotely on a downstream database.*

- *A single database can have one or more capture processes that capture local changes and other capture processes that capture changes from a remote source database. That is, you can configure a single database to perform both local capture and downstream capture.*

Local Capture

- *The Source Database Performs All Change Capture Actions*
  - The DBMS_CAPTURE_ADM.BUILD procedure is run to extract (or build) the data dictionary to the redo log.
  - Supplemental logging at the source database places additional information in the redo log. This information might be needed when captured changes are applied by an **apply process**.
  - The first time a capture process is started at the database, Oracle uses the extracted data dictionary information in the redo log to create a **LogMiner data dictionary**, which is separate from the primary data dictionary for the source database. Additional capture processes can use this existing LogMiner data dictionary, or they can create new LogMiner data dictionaries.
  - A capture process scans the redo log for changes using LogMiner.

Local Capture

- *The Source Database Performs All Change Capture Actions*
  - The **rules engine** evaluates changes based on the **rule**s in one or more of the capture process **rule set**s.
  - The capture process enqueues changes that satisfy the rules in its rule sets into a local ANYDATA queue.
  - If the captured changes are shared with one or more other databases, then one or more **propagation**s propagate these changes from the source database to the other databases.
  - If database objects at the source database must be instantiated at a **destination database**, then the objects must be prepared for **instantiation** and a mechanism such as an Export utility must be used to make a copy of the database objects.

Local Capture

- *Advantages:*
  - Configuration and administration of the capture process is simpler than when downstream capture is used. When you use local capture, you do not need to configure redo log file copying to a **downstream database**, and you administer the capture process locally at the database where the captured changes originated.

  - A **local capture process** can scan changes in the online redo log before the database writes these changes to an archived redo log file. When you use downstream capture, archived redo log files are copied to the downstream database after the source database has finished writing changes to them, and some time is required to copy the redo log files to the downstream database.

  - The amount of data being sent over the network is reduced, because the entire redo log file is not copied to the downstream database. Even if **captured message**s are propagated to other databases, the captured messages can be a subset of the total changes made to the database, and only the LCRs that satisfy the rules in the rule sets for a **propagation** are propagated.

Local Capture

- *Advantages:*
  - Security might be improved because only the source (local) database can access the redo log files. For example, if you want to capture changes in the hr schema only, then, when you use local capture, only the source database can access the redo log to enqueue changes to the hr schema into the capture process **queue**. However, when you use downstream capture, the redo log files are copied to the downstream database, and these redo log files contain all of the changes made to the database, not just the changes made to the hr schema.

  - Some types of **custom rule-based transformation**s are simpler to configure if the capture process is running at the local source database. For example, if you use local capture, then a custom rule-based transformation can use cached information in a PL/SQL session variable which is populated with data stored at the source database.

  - In a Streams environment where messages are captured and applied in the same database, it might be simpler, and use fewer resources, to configure local queries and computations that require information about captured changes and the local data.

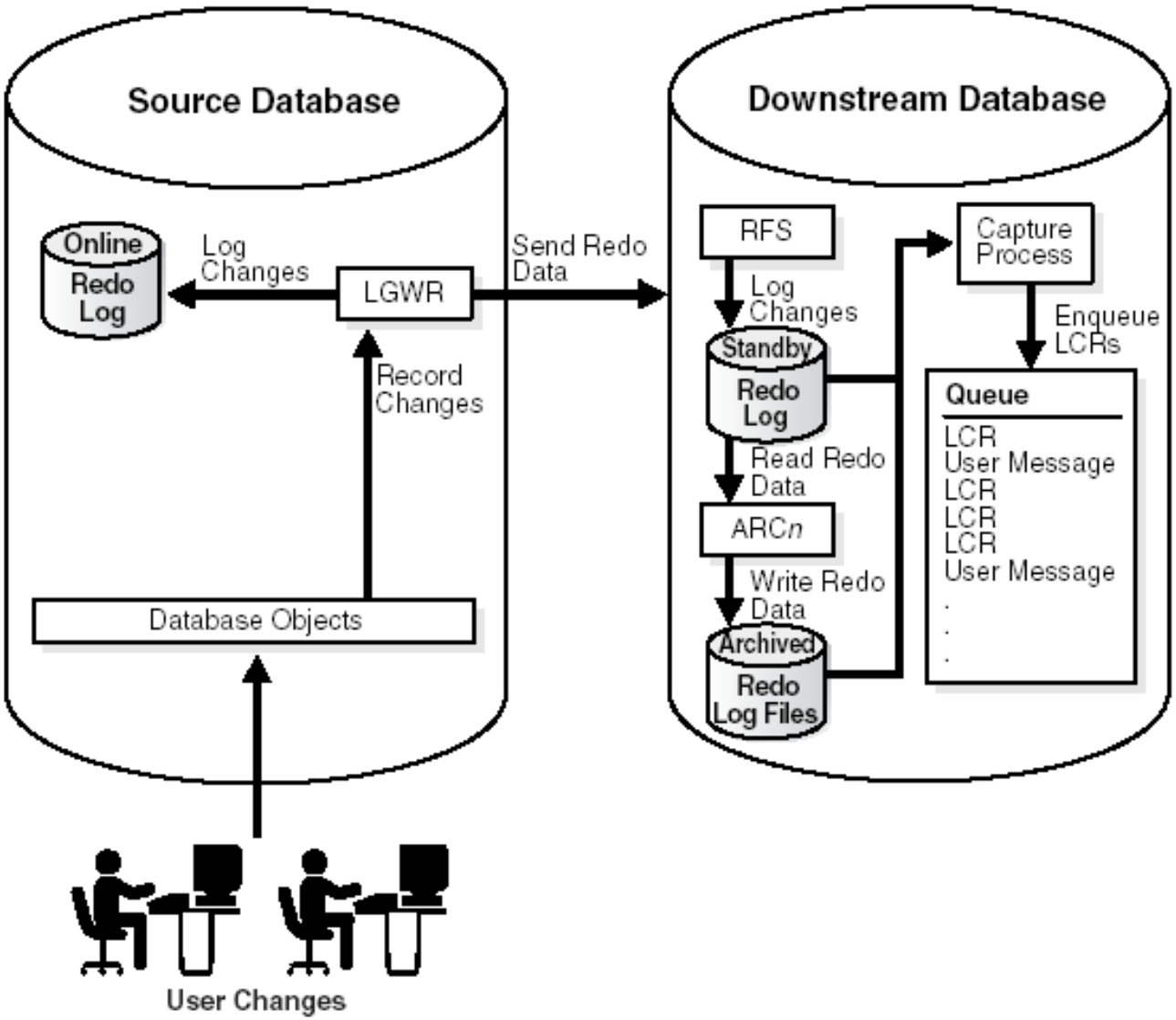
Downstream Capture

- Downstream capture means that a capture process runs on a database other than the **source database**. The following types of downstream capture configurations are possible:
  - real-time downstream capture
  - archived-log downstream capture
- A real-time downstream capture process and one or more archived-log downstream capture processes can coexist at a **downstream database**.

Real-Time Downstream Capture

- *A **real-time downstream capture** configuration works in the following way:*
  - Redo transport services use the log writer process (LGWR) at the source database to send redo data to the downstream database either synchronously or asynchronously. At the same time, the LGWR records redo data in the online redo log at the source database.
  - A remote file server process (RFS) at the downstream database receives the redo data over the network and stores the redo data in the standby redo log.
  - A log switch at the source database causes a log switch at the downstream database, and the ARCH*n* process at the downstream database archives the current standby redo log file.
  - The real-time downstream capture process captures changes from the standby redo log whenever possible and from the archived standby redo log files whenever necessary. A capture process can capture changes in the archived standby redo log files if it falls behind. When it catches up, it resumes capturing changes from the standby redo log.

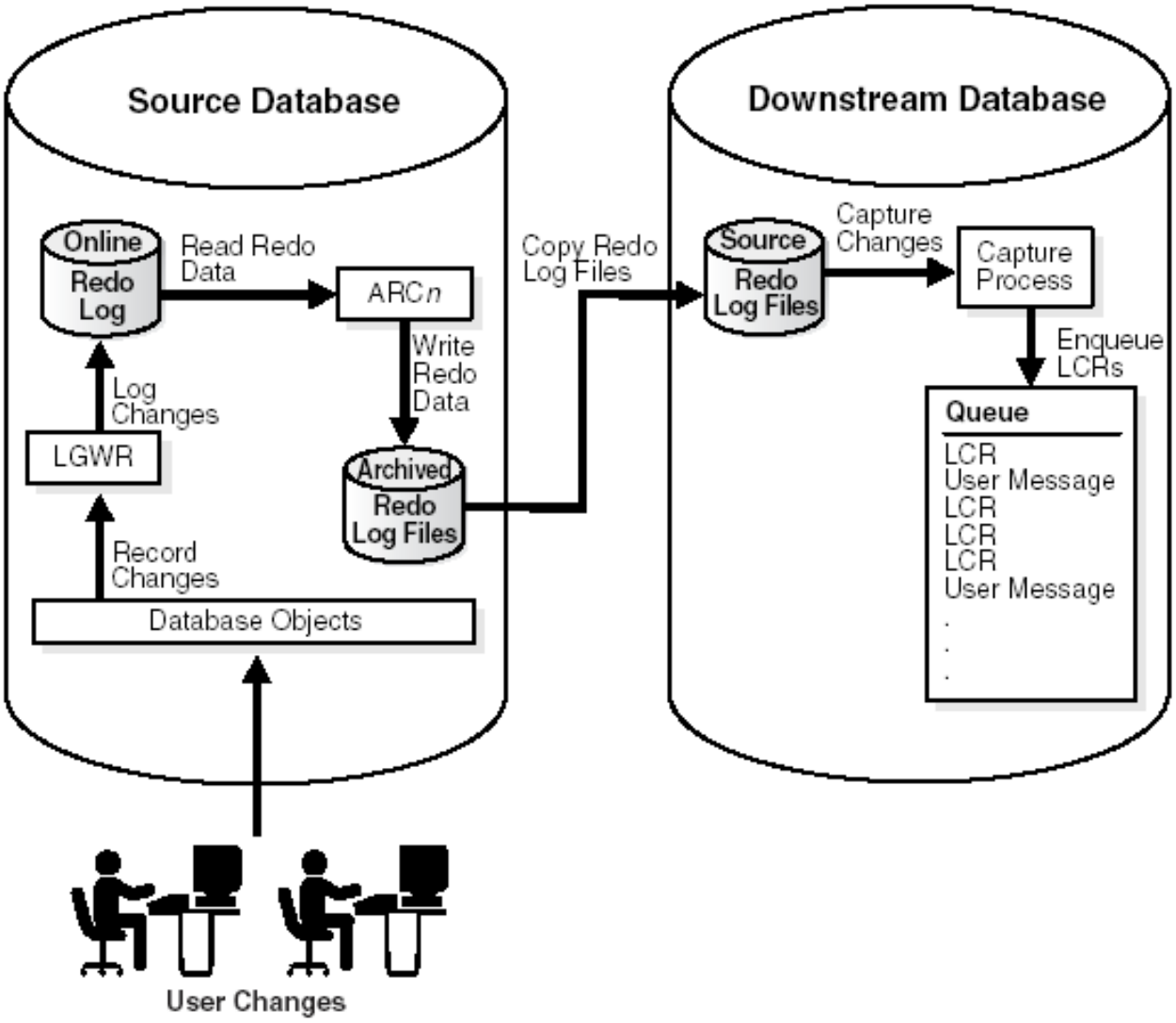
Real-Time Downstream Capture



Archived-Log Downstream Capture

- *A **archived-log downstream capture process** configuration means that* :
  - archived redo log files from the source database are copied to the downstream database
  - the capture process captures changes in these archived redo log files
  - You can copy the archived redo log files to the downstream database using redo transport services, the BMS_FILE_TRANSFER package, file transfer protocol (FTP), or some other mechanism

Archived-Log Downstream Capture



Downstream Capture

Advantages of Downstream Capture

- Capturing changes uses fewer resources at the source database because the downstream database performs most of the required work.
- If you plan to capture changes originating at multiple source databases, then capture process administration can be simplified by running multiple archived-log downstream capture processes with different source databases at one downstream database. That is, one downstream database can act as the central location for change capture from multiple sources. In such a configuration, one real-time downstream capture process can run at the downstream database in addition to the archived-log downstream capture processes.
- Copying redo data to one or more downstream databases provides improved protection against data loss. For example, redo log files at the downstream database can be used for recovery of the source database in some situations.

Department of Computer Science and Technology, Peking University, Nov , 2006