IBM Informix Dynamic Server (IDS)

IDS 11* Technical Features/Enhancements Overview

Jan Musil

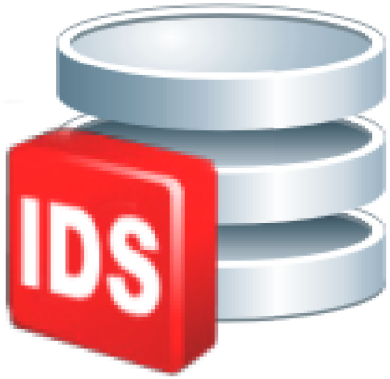
jan_musil@cz.ibm.com

* Includes version 11.10 and 11.50[5]

12 January 2010

Content

- **Administration and Usability**
- **Business Continuity**
- **Performance Enhancements**
- **Application Development**
- **Q & A**



2

Technical Overview

- **Administration and Usability**
- **Business Continuity**
- **Performance Enhancements**
- **Application Development**

3

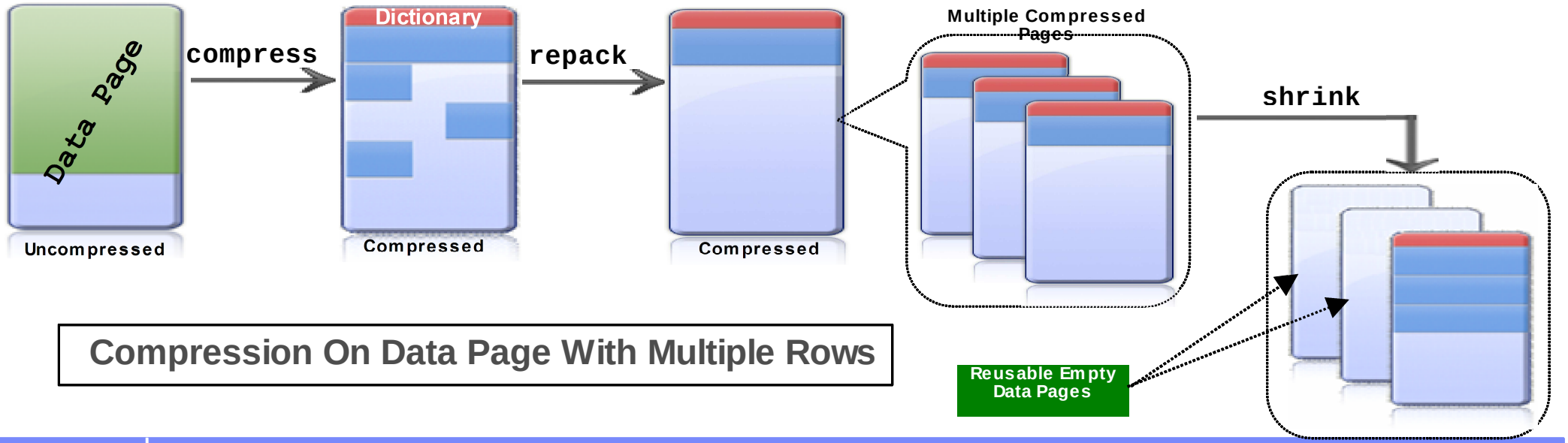
Administration and Usability

- **Compression and Storage Optimization**
- **Dynamically Start, Stop, or Restart Listen Threads**
- **Forcing the Database Server to Shut Down**

4

Compression and Storage Optimization

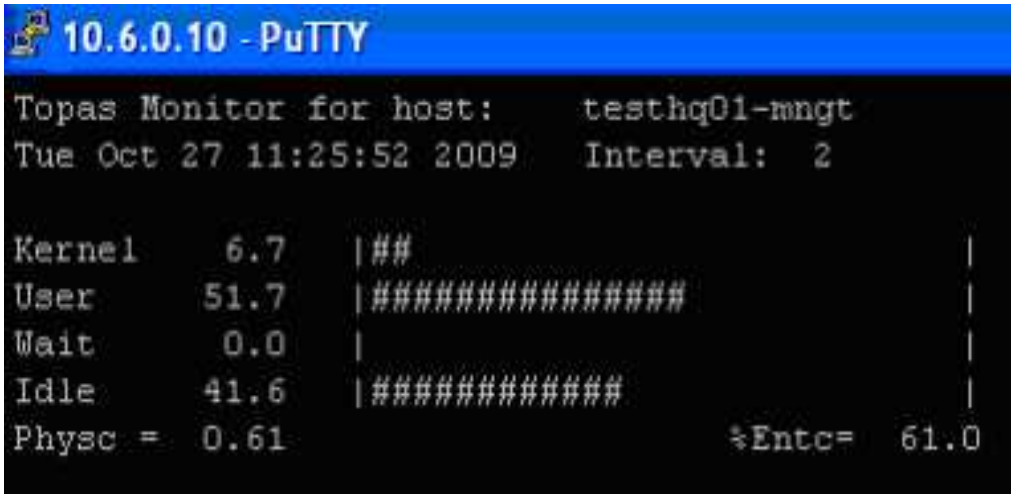
- Row base compression
- Estimation tools helps in finding compression ratio
- Compression can save 40-50% of the db storage requirements
- For IO-bound workloads compression also improves performance
- Significant memory savings – more efficient memory utilization
- Also save on backup storage and disaster recovery storage



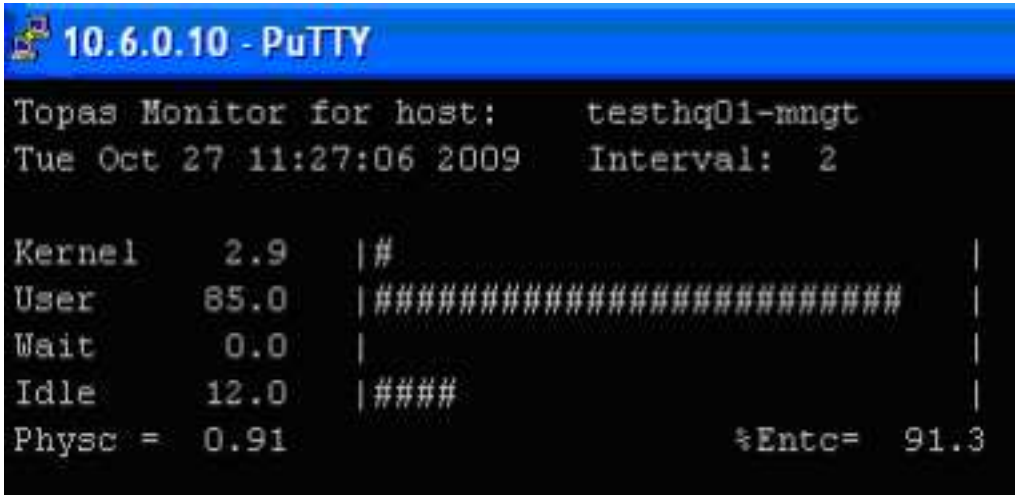
Compression On Data Page With Multiple Rows

Compression and Storage Optimization

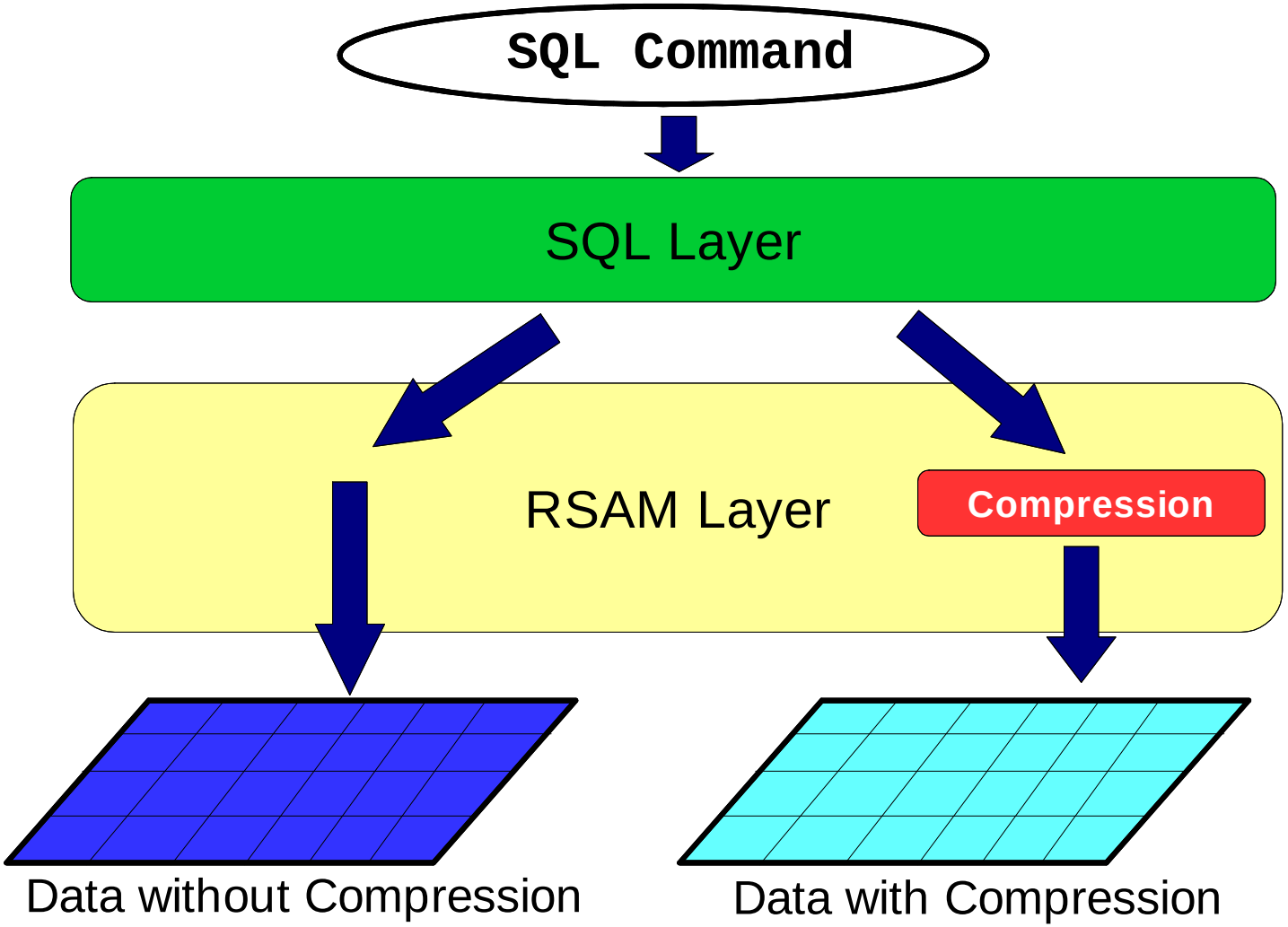
- No Compression
  - SQL Query: **real 0m28.514s**



- After Compression
  - SQL Query: **real 0m16.110s**



Compression and Storage Optimization



- Application
- ER replication
- InfoSphere CDC

- MACH-11
  - HDR
  - RSS

Dynamically Start, Stop, or Restart Listen Threads (1)

- You can dynamically start, stop, or stop and start a listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections
- For example, you might want to stop listen threads that are unresponsive and then start new ones in situations when other server functions are performing normally and you do not want to shut down the server
- Prerequisite
  - The listen thread must be defined in the sqlhosts file for the server
  - If necessary, before start, stop, or restart a listen thread, you can revise the sqlhosts entry

Dynamically Start, Stop, or Restart Listen Threads (2)

- To dynamically start, stop, or restart listen threads run one of the following onmode -P commands:
  - onmode -P start server_name
  - onmode -P stop server_name
  - onmode -P restart server_name
- Alternatively using task() or admin() commands
  - EXECUTE FUNCTION task("start listen", "server_name");
  - EXECUTE FUNCTION task("stop listen" ,"server_name");
  - EXECUTE FUNCTION task("restart listen", "server_name");

Dynamically Start, Stop, or Restart Listen Threads (3)

- For example, either of the following commands starts a new listen thread for a server named ids_serv2:
  - onmode -P start ids_serv2
  - EXECUTE FUNCTION task("start listen", "ids_serv2");
- ids_serv2 onsoctcp 10.0.0.56 20010

Forcing the Database Server to Shut Down (1)

- Use the **onclean** utility to force a shut down of the database server when normal shut down with the *onmode* utility fails or when you cannot restart the server.
- The onclean utility attempts to clean up shared memory, semaphores, and stops database server virtual processes.

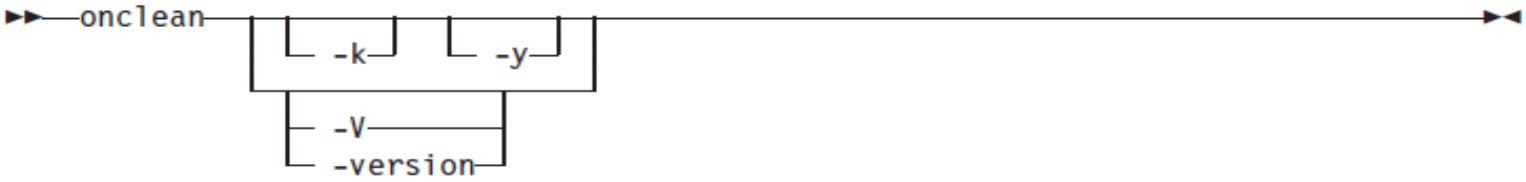
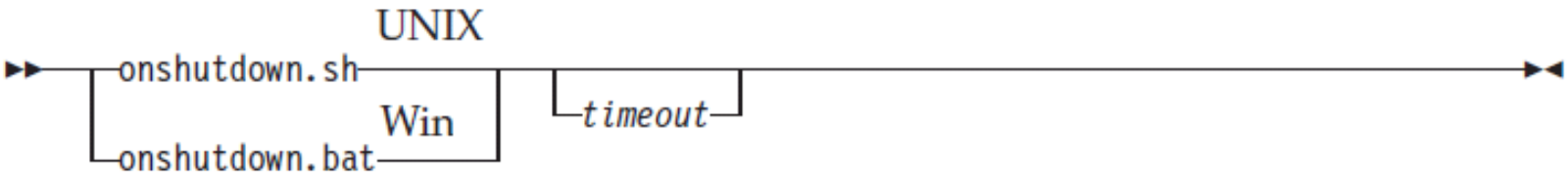


Table 9-1. Syntax Elements of the **onclean** *Command*

Element	Purpose
-k	Shuts down a server that is online by stopping database server virtual processes and attempting to clean up the remaining semaphores and shared-memory segments, even if they are still running.
-V	Displays short version information.
-version	Displays full version information.
-y	Does not prompt for confirmation.

Forcing the Database Server to Shut Down (2)

- You can automate shutting down the database server with the onshutdown script, which calls the onclean -ky command if necessary
- Use the onshutdown script to automate shutting down the database server by attempting to shut down the server normally, and then if the server has not shut down after a specified time, forcing the server to shut down.



Element	Purpose
timeout	The number of seconds after the **onmode -ky** command has been run before running the **onclean -ky** command. Must be a positive integer from 10 to 60. The default value is 30 seconds.

Forcing the Database Server to Shut Down (3)

- !! Use the onclean utility with caution !!
  - When you run onclean, any pending transactions and processes fail to complete, and user sessions are disconnected abruptly.
  - However, the database server rolls back transactions when it restarts.
- !! Use the onshutdown script with caution !!
  - If the script needs to run the onclean -ky command, any pending transactions and processes fail to complete, and user sessions are disconnected abruptly
  - However, the database server rolls back transactions when it restarts.

Technical Overview

- Administration and Usability
- **Business Continuity**
- Application Development
- Performance Enhancements

14

Business Continuity

- **Configuring RS Secondary Server Latency for Disaster Recovery**
- **External Backup in RSS**
- **Connection Manager Proxy Support**

Configuring RS Secondary Server Latency for Disaster Recovery (1/2)

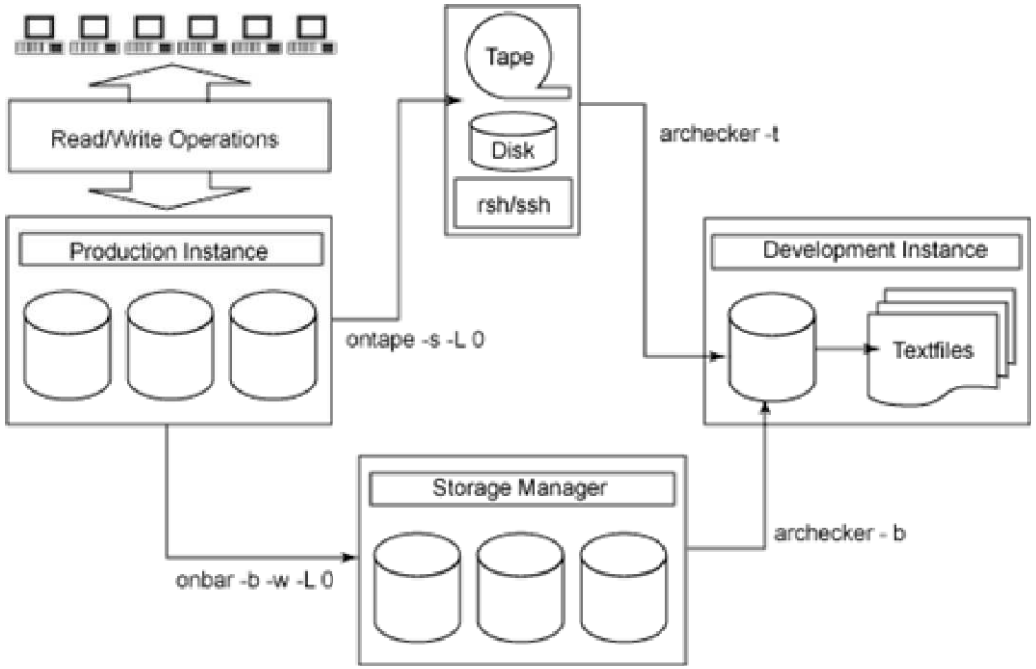
- To aid in disaster recovery scenarios, you can now configure RS secondary servers to wait for a specified period of time before applying logs.
- Delaying the application of log files allows you to recover quickly from erroneous database modifications by restoring from the RS secondary server.

Configuring RS Secondary Server Latency for Disaster Recovery (2/2)

- LOG_STAGING_DIR
  - Use the LOG_STAGING_DIR configuration parameter to specify the location of log files received from the primary server when configuring delayed application of log files on RS secondary servers.
- DELAY_APPLY
  - Use the DELAY_APPLY configuration parameter to configure RS secondary servers to wait for a specified period of time before applying logs.
- STOP_APPLY
  - Use the STOP_APPLY configuration parameter to stop an RS secondary server from applying log files received from the primary server.

External Backup in RSS

- It allows the DBA to create an External backup from a RSS server, so that the primary is freed of this task.
- During the archive the primary server is totally free and can act independently about the archive (The primary server is not blocked).
- During the archive the RSS server is in STOP_APPLY mode:
  - Immediately after the archive checkpoint is processed, the RSS stops applying logical logs, however it keeps receiving them from the primary to avoid hangs in the primary.



Connection Manager Proxy Support

- The Connection Manager can now be configured as a proxy server when clients connect to Informix data servers from outside a firewall.
- A new oncmsm configuration variable has been added, called MODE. You can specify either REDIRECT or PROXY mode for each SLA.
- In PROXY mode the client stays connected to the Connection Manager which then relays all the client traffic to the server.
- For performance reasons it is recommended to have multiple Connection Managers when operating in PROXY mode.

Technical Overview

- Administration and Usability
- Business Continuity
- **Performance Enhancements**
- Application Development

20

Performance Enhancements

- **Direct I/O and Concurrent I/O support**
- **Data Scan Enhancements**

21

Direct I/O for cooked dbspace chunks on UNIX

- IDS allows you to use either raw devices or cooked files for dbspace chunks
- Cooked files are slower due to additional overhead and buffering provided by the file system
- Direct I/O bypasses the use of the file system buffers, hence is more efficient for reads and writes that go to disk
- Direct I/O improves the performance of cooked files used for dbspace chunks
- Direct I/O can be enabled with the DIRECT_IO configuration parameter

If your file system supports direct I/O, performance for cooked files can approach the performance of raw devices used for dbspace chunks

Concurrent I/O on AIX

- Applies only to cooked chunk files
- Benefits:
  - Uses direct I/O (no file system buffering)
  - Avoids unnecessary serialization of I/O
  - Higher I/O rates, especially when data striped across multiple disks
- Enable by setting DIRECT_IO onconfig parameter to 2
- "onstat -d" shows status
  - "C" for concurrent I/O
  - "D" for direct I/O

Data Scan Enhancements

- Improves warehouse performance by
  - Enabling light scans on tables with rows that can span pages, which includes:
    - Varchar, lvarchar, nvarchar
    - Compressed tables
    - Any table with rows longer than a page
- Light Scan for fixed length rows already enabled
- ONCONFIG file:
  - BATCHEDREAD_TABLE 1
- Shell environment (set in IDS shell before starting):
  - Csh family: IFX_BATCHEDREAD_TABLE 1
  - Ksh family: export IFX_BATCHEDREAD_TABLE=1
- Single session:
  - set environment IFX_BATCHEDREAD_TABLE ‘1’;

Technical Overview

- Administration and Usability
- Business Continuity
- Performance Enhancements
- **Application Development**

Application Development

- **Loading data with MERGE statement**
- **Update and Delete with MERGE statement**
- **External Table**
- **Attach/Detach Fragment Optimization**
- **Retrieving data with Hierarchical queries**
- **Rolling Back SQL Transactions to a Savepoint**

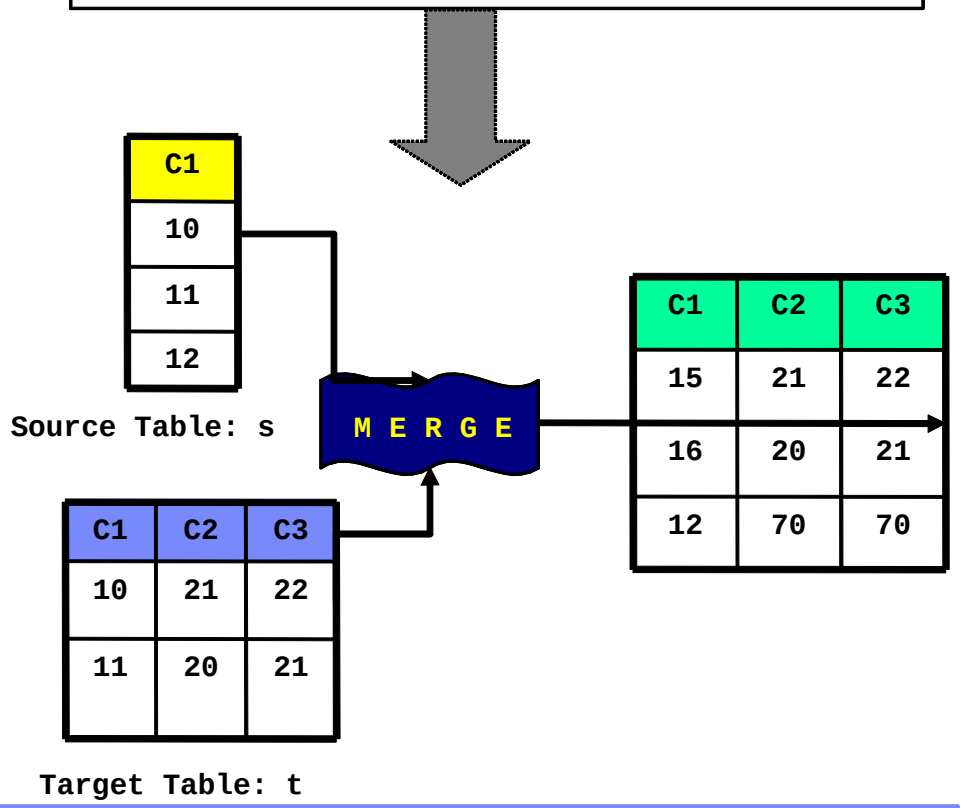
Cont'd …

26

Loading data with MERGE statement

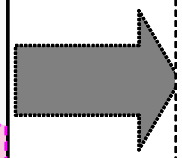
- Transfers data to target table using UPSERT
  - UPSERT = a combination of INSERT and UPDATE in one statement
- Replaces current UPSERT logic implemented in client applications.
- Extension of ANSI/ISO 2003 standard.
- Key components of ETL (ELT)* in data warehouse environments.
- Merges rows to a target table based on a condition.
  - TRUE: Update the target table row.
  - FALSE: Insert the target table row

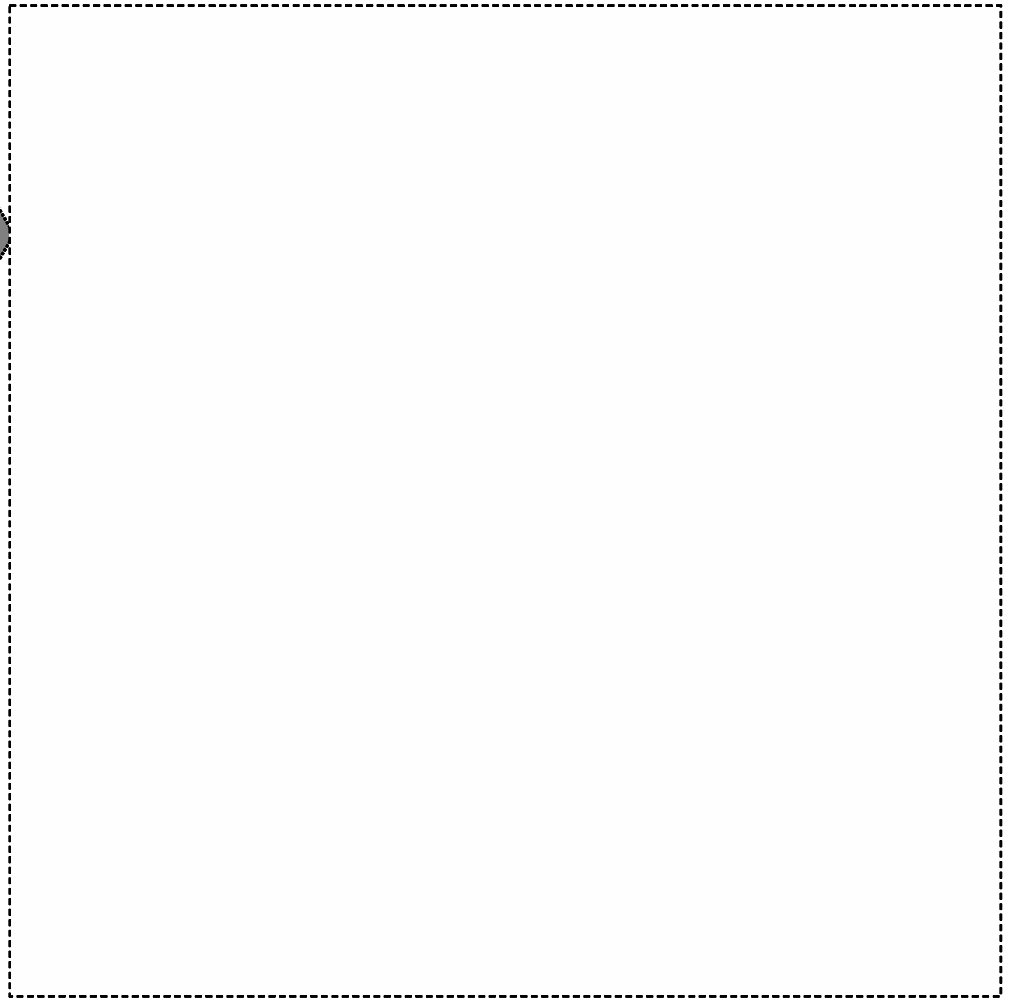
```
MERGE
  INTO TARGET t
  USING SOURCE s
  ON t.c1=s.c1
  WHEN MATCHED THEN
    UPDATE SET t.c1=t.c1+5
  WHEN NOT MATCHED
    THEN INSERT (t.c1, t.c2, t.c3)
    values (s.c1, 70, 70);
```



Update and Delete Joins with MERGE statement

```
MERGE INTO bonuses D
USING employee S
ON D.employee_id = S.employee_id
WHEN MATCHED THEN DELETE;

Optional: "WHEN NOT MATCHED"
```



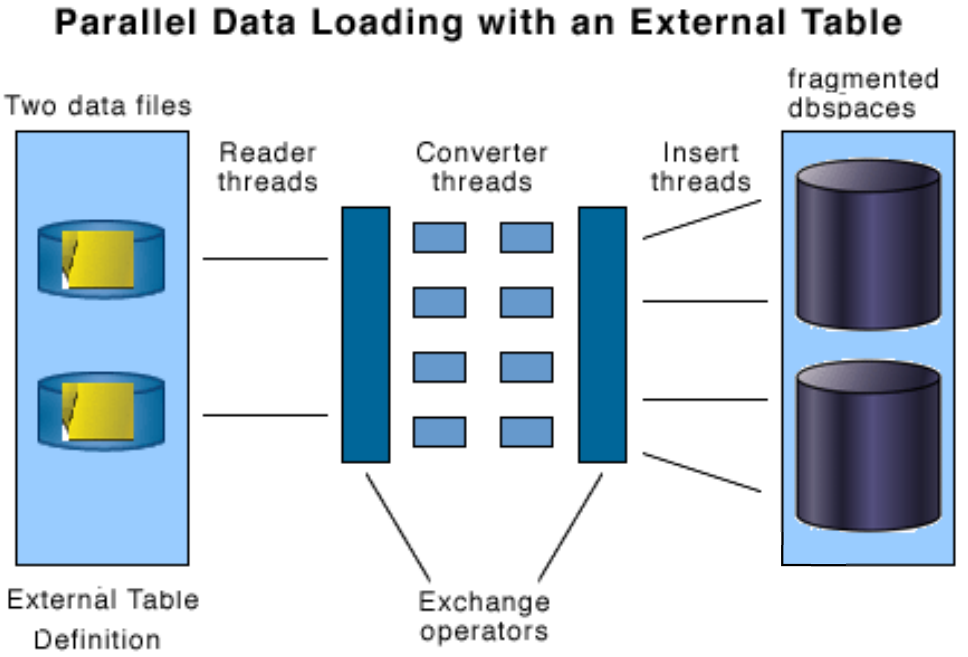


Source table may be a table expression consisting of joins

* C1 -> employee_id, C2 -> salary

External Tables

- **Performance**
  - Faster load and unload of massive data sets.
    - Internal tests shows 2x for unload and 3x for load of large tables over HPL.



- **Ease of Use**
  - Syntactically, an external table can be used in an SQL statement in place of a regular table.
    - No coding needed compared to use of VTI.
  - External table can be used in Stored procedure for load and unload
  - No need of DBA privilege to do Load/Unload using external tables
- **Most requested feature for XPS users**

External Table Examples (1)

- An external table named *empdata* is created with two columns.
- The DATAFILES clause indicates
  - the location of the data file
  - specifies that the file is delimited
  - indicates the location of the reject file
  - indicates that the reject file can contain no more than 100 errors
  - and that data is to be loaded using deluxe mode.

```
CREATE EXTERNAL TABLE empdata
(
   empname char(40),
   empdoj date
)
USING
(DATAFILES
    (
        "DISK:/work/empdata.unl"
    ),
    FORMAT "DELIMITED",
    REJECTFILE "/work/errlog/empdata.rej",
    MAXERRORS 100,
    DELUXE
);
```

External Table Examples (2)

- The following example uses the column names and data types of the *empdata* table and uses them for the external table.

```
CREATE EXTERNAL TABLE emp_ext SAMEAS empdata
USING
(DATAFILES
    (
        "DISK:/work/empdata2.unl"
    ),
    REJECTFILE "/work/errlog/empdata2.rej",
    DELUXE
);
```

External Table Examples (3)

- The following example shows statements used to load data from a database table into an external table.

```
CREATE EXTERNAL TABLE ext1( col1 int )
 USING
 (DATAFILES
      (
         "DISK:/tmp/ext1.unl"
      )
);

CREATE TABLE base (col1 int);
INSERT INTO ext1 SELECT * FROM base;
```

- The SELECT...INTO EXTERNAL syntax can be used to unload data as in the following example.

```
SELECT * FROM base
INTO EXTERNAL emp_target
 USING
 (DATAFILES
      (
           "DISK:/tmp/ext1.unl"
      )
);
```

External Table Examples (4)

- The following example selects from an external table and shows various ways to load external data into a database table.

```
CREATE EXTERNAL TABLE ext1( col1 int )
  USING
  (DATAFILES
      (
          "DISK:/tmp/ext1.unl"
      )
);

CREATE TABLE target1 (col1 int);
CREATE TABLE target2 (col1 serial8, col2 int);

SELECT * FROM ext1;
SELECT col1,COUNT(*) FROM ext1 GROUP BY 1;
SELECT MAX(col1) FROM ext1;
SELECT col1 FROM ext1 a, systables b WHERE a.col1=b.tabid;

INSERT INTO target1 SELECT * FROM ext1;
INSERT INTO target2 SELECT 0,* FROM ext1;
```

External Table Examples (5)

- The next example creates an external table named *emp_ext*, defines the column names and data types, and unloads the data from the database using fixed format.

```
CREATE EXTERNAL TABLE emp_ext
 ( name CHAR(18) EXTERNAL CHAR(20),
   address VARCHAR(40) EXTERNAL CHAR(40),
   empno INTEGER EXTERNAL CHAR(6)
  )
USING (
  FORMAT 'FIXED',
  DATAFILES
     (
        "DISK:/work2/mydir/emp.fix"
     )
);

INSERT INTO emp_ext SELECT * FROM employee;
```

External Table Examples (6)

- The next example creates an external table named *emp_ext* and loads data into the database from a fixed format file.

```
CREATE EXTERNAL TABLE emp_ext
 ( name CHAR(18) EXTERNAL CHAR(18),
   address VARCHAR(40) EXTERNAL CHAR(40),
   empno INTEGER EXTERNAL CHAR(6)
 )
USING (
  FORMAT 'FIXED',
  DATAFILES
     (
         "DISK:/work2/mydir/emp.fix"
     )
);

INSERT INTO employee SELECT * FROM emp_ext;
```

Attach/Detach Fragment Optimization (1/2)

- Currently, an exclusive lock on the base table is required to add/drop a fragment
- Optimization is for the system to force users off by aborting transactions
- Connections are still intact
- Lock mode wait can be set to wait certain amount of time before “Alter Fragment” statement
- An ALTER FRAGMENT operation can now force out transactions to get exclusive access to tables
- Allows programmatic attaching/detaching fragments

Attach/Detach Fragment Optimization (2/2)

```
> BEGIN;
Started transaction.
> INSERT INTO batch_post_id VALUES (0, CURRENT, 'Week End');
1 row(s) inserted.
```

```
> ALTER FRAGMENT ON TABLE batch_post_id INIT
> FRAGMENT BY ROUND ROBIN IN dbs1, dbs2;
  242: Could not open database table (cosmo.batch_post_id).
  113: ISAM error:  the file is locked.
Error in line 1
Near character position 78
> SET ENVIRONMENT FORCE_DDL_EXEC '10';
Environment set.
> ALTER FRAGMENT ON TABLE batch_post_id INIT
> FRAGMENT BY ROUND ROBIN IN dbs1, dbs2;
Alter fragment completed.
```

```
> COMMIT;
  458: Long transaction aborted.
12204: RSAM error: Long transaction detected.
```

Retrieving data with Hierarchical queries

- Retrieves relational hierarchal data with parent-child dependencies
- Implement using “START WITH ... CONNECT BY” directive in SELECT clause
- Syntax:

- Comparison with Node Server Extension

Node Server Extension	CONNECT BY
Depth for hierarchy is limited to 16	Depth is (almost) unlimited (as much as fragmented temp table can hold)
Queries use normal relational operators to query hierarchy	Queries use recursive operators to query hierarchy

Retrieving data with Hierarchical queries

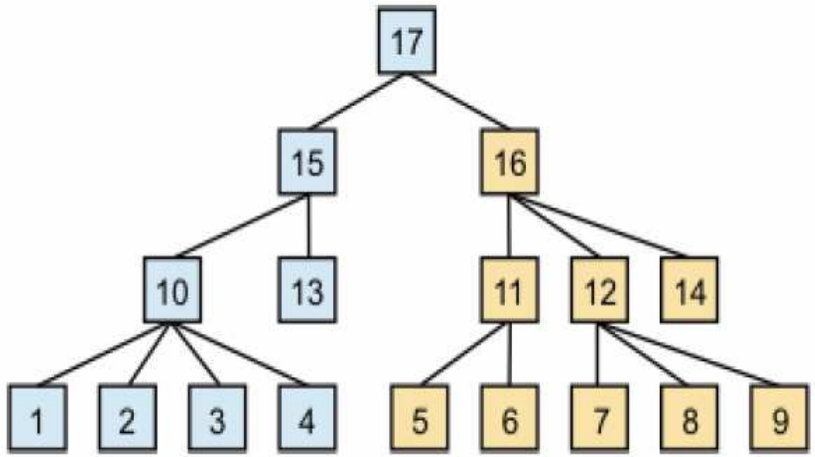
```
CREATE TABLE employee(
                        empid  INTEGER NOT NULL PRIMARY KEY,
                        name   VARCHAR(10),
                        salary DECIMAL(9, 2),
                        mgrid  INTEGER
);

INSERT INTO employee VALUES  ( 1, 'Jones',    30000, 10);
INSERT INTO employee VALUES  ( 2, 'Hall',     35000, 10);
INSERT INTO employee VALUES  ( 3, 'Kim',      40000, 10);
INSERT INTO employee VALUES  ( 4, 'Lindsay',  38000, 10);
INSERT INTO employee VALUES  ( 5, 'McKeough', 42000, 11);
INSERT INTO employee VALUES  ( 6, 'Barnes',   41000, 11);
INSERT INTO employee VALUES  ( 7, 'O''Neil',  36000, 12);
INSERT INTO employee VALUES  ( 8, 'Smith',    34000, 12);
INSERT INTO employee VALUES  ( 9, 'Shoeman',  33000, 12);
INSERT INTO employee VALUES  (10, 'Monroe',   50000, 15);
INSERT INTO employee VALUES  (11, 'Zander',   52000, 16);
INSERT INTO employee VALUES  (12, 'Henry',    51000, 16);
INSERT INTO employee VALUES  (13, 'Aaron',    54000, 15);
INSERT INTO employee VALUES  (14, 'Scott',    53000, 16);
INSERT INTO employee VALUES  (15, 'Mills',    70000, 17);
INSERT INTO employee VALUES  (16, 'Goyal',    80000, 17);
INSERT INTO employee VALUES  (17, 'Urbassek', 95000, NULL);
```

Retrieving data with Hierarchical queries

```
SELECT empid, name, mgrid, CONNECT_BY_ISLEAF leaf, CONNECT_BY_ISCYCLE cycle
FROM   employee
START WITH name = 'Goyal'
CONNECT BY NOCYCLE PRIOR empid = mgrid;
```

empid	name	mgrid	leaf	cycle
16	Goyal	17	0	0
14	Scott	16	1	0
12	Henry	16	0	0
9	Shoeman	12	1	0
8	Smith	12	1	0
7	O'Neil	12	1	0
11	Zander	16	0	0
6	Barnes	11	1	0
5	McKeough	11	0	0
17	Urbassek	5	0	1
15	Mills	17	0	0
13	Aaron	15	1	0
10	Monroe	15	0	0
4	Lindsay	10	1	0
3	Kim	10	1	0
2	Hall	10	1	0
1	Jones	10	1	0

17 row(s) retrieved.



Rolling Back SQL Transactions to a Savepoint (1/3)

- A savepoint identifies an arbitrary location within the statements of an SQL transaction.
- Within its transaction, the savepoint resembles a statement label to which the ROLLBACK statement of SQL can cancel any changes to the database that the transaction produced between the savepoint and the ROLLBACK statement.
- A client application can implement error-handling logic that rolls back only the portion of the transaction, rather than cancelling the entire transaction if an exception occurs.

Rolling Back SQL Transactions to a Savepoint (2/3)

- The following new SQL statements are implemented for savepoints
  - The SAVEPOINT statement creates a savepoint within the current SQL transaction.
  - The RELEASE SAVEPOINT statement destroys a specified savepoint, as well as any other savepoints that exist between the RELEASE statement and the savepoint that it references.
  - The ROLLBACK WORK TO SAVEPOINT statement discards changes to the schema of the database or to its data values by statements that follow the savepoint, but the effects of DDL and DML statements that preceded the savepoint persist.

Rolling Back SQL Transactions to a Savepoint (3/3)

```
BEGIN WORK;
CREATE DATABASE third_base IN db3 WITH BUFFERED LOG;
SAVEPOINT sp46;
CREATE TABLE tab1 ( col1 INT, col2 CHAR(24));
SAVEPOINT sp45 UNIQUE;
...
CREATE TABLE tab2 ( col1 INT8, col2 LVARCHAR(24000));
SAVEPOINT sp44;
...
RELEASE SAVEPOINT sp45;
ROLLBACK TO SAVEPOINT;
```

- RELEASE SAVEPOINT statement destroys two savepoints, sp45 and sp44
- The only remaining savepoint in the current savepoint level is sp46
- ROLLBACK TO SAVEPOINT statement cancels the DDL statements that created tab1 and tab2
- The rollback does not, however, cancel the CREATE DATABASE statement that created the third_base database

Otázky a odpovědi

Děkuji za pozornost!

Jan Musil
jan_musil@cz.ibm.com