Oracle 11g New Features for DBAs

Arup Nanda

Agenda

- Tons of new features in 11g
- It's not "new" anymore. Plenty of material available – blogs, articles, books

- Compelling reasons for upgrade. Biggest bang for the buck.
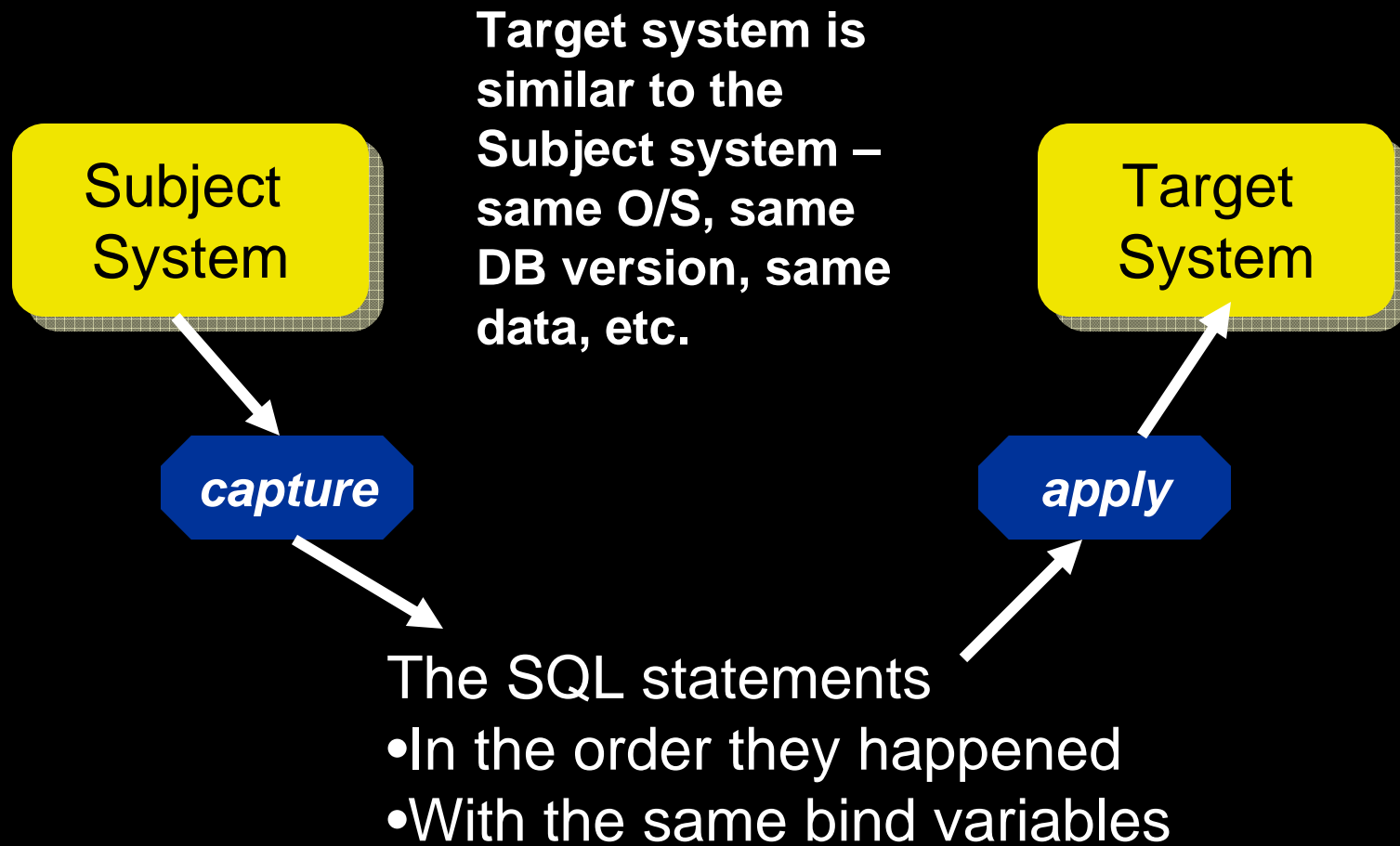- Exclusively for DBAs; not Developers

Coverage

- Only the most valuable features
- Stress on "how to use", rather than syntax
- Companion material – "*Oracle Database 11g: The Top New Features for DBAs and Developers*" on OTN
- http://www.oracle.com/technology/pub/articles/oracle-database-11g-top-features/index.html
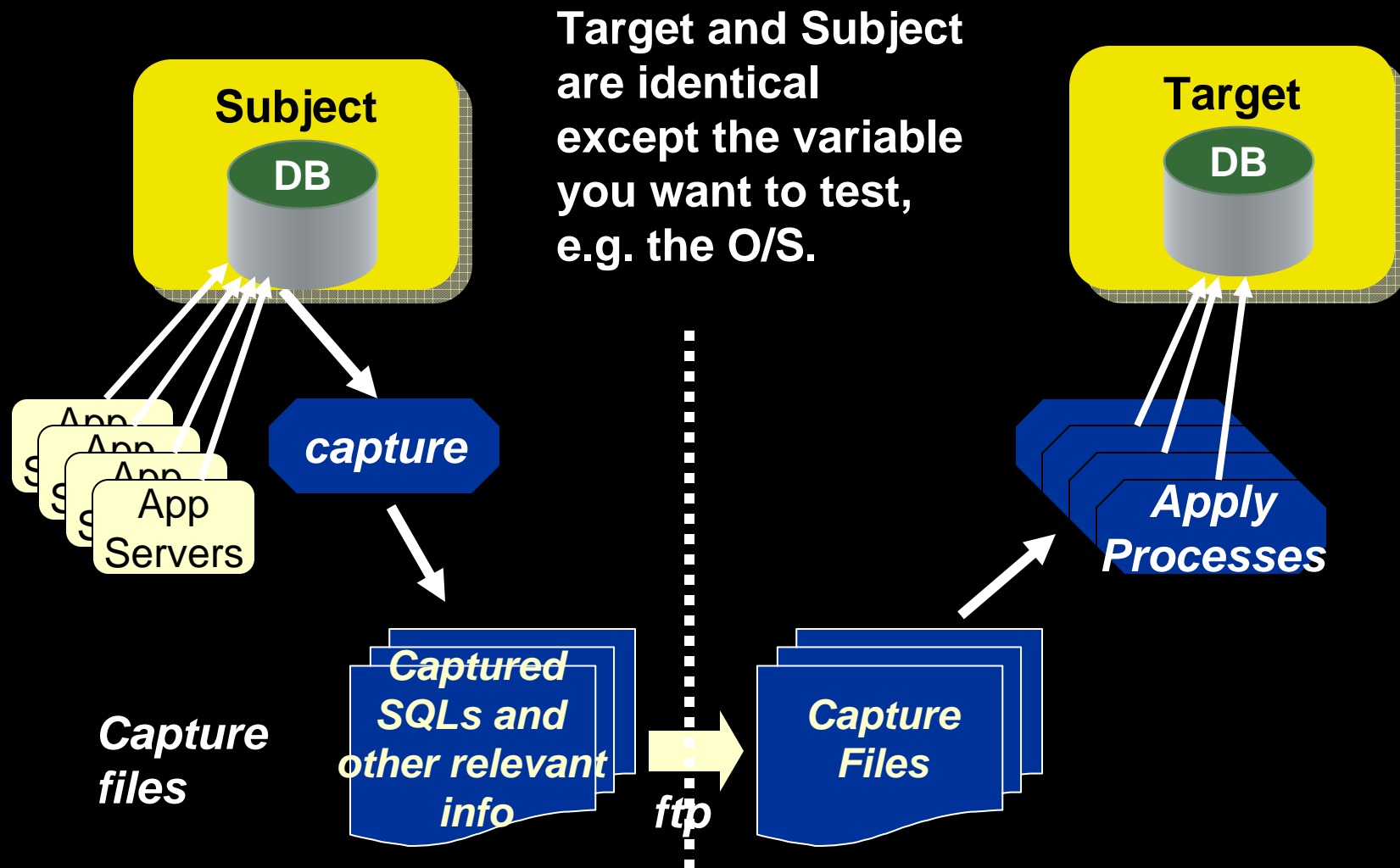- It has complete syntax, working examples and videos (coming)

Database Replay

- Change is the *only* constant
  - What happens when you change something – init params, storage, O/S, kernel params …
- There are always risks of a change
- You can mitigate by subjecting the changed system to the very similar workload and comparing the results
- The keyword is "*similar* workload"
- Load generators do not have the fidelity

A True Test

Subject System

Target system is similar to the Subject system – same O/S, same DB version, same data, etc.

Target System

capture

apply

The SQL statements
•In the order they happened
•With the same bind variables

Database Replay Concepts

Subject
DB

Target and Subject are identical except the variable you want to test, e.g. the O/S.

Target
DB

App
App
App
App
Servers

capture

Apply Processes

Capture files

Captured SQLs and other relevant info

ftp

Capture Files

6

Case: Changing a Database Parameter

- You can test the effects on the Subject DB
  - Enable flashback for DB
  - Capture workload for, say, a week
  - Stop apps, create a savepoint
  - Change the parameter
  - Replay the captured workload
  - Compare the results
  - Decide to keep the param or not
  - Flashback database to savepoint
  - Start the apps

Effect of Moving to ASM

You want to see if moving to ASM would help

1. Setup standby on ASM from the Filesystem DB
2. Capture workload
3. Stop apps
4. Sync up standby and break
5. Update pfile of ASM DB the same as the FS DB
6. Shutdown FS DB
7. Replay workload

FS

Server

standby

ASM
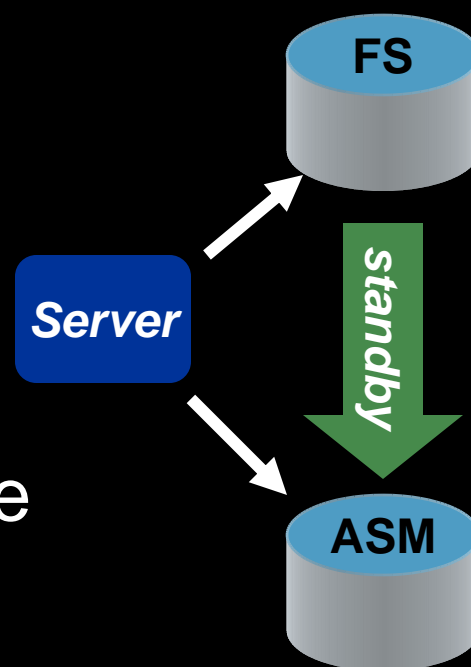
Upgrades from 10g to 11g

- 10.2.0.4 patchkit actually has the Database Replay tools built in

- `DBMS_WORKLOAD_CAPTURE` package available.

- You can use this to capture workload from 10.2.0.4 and apply those to a 11.1 DB.

- http://download.oracle.com/docs/cd/B19306_01/server.102/e12024/toc.htm

- MetaLink Note 560977.1 shows the one-off patches available for all other releases to enable capture of workload
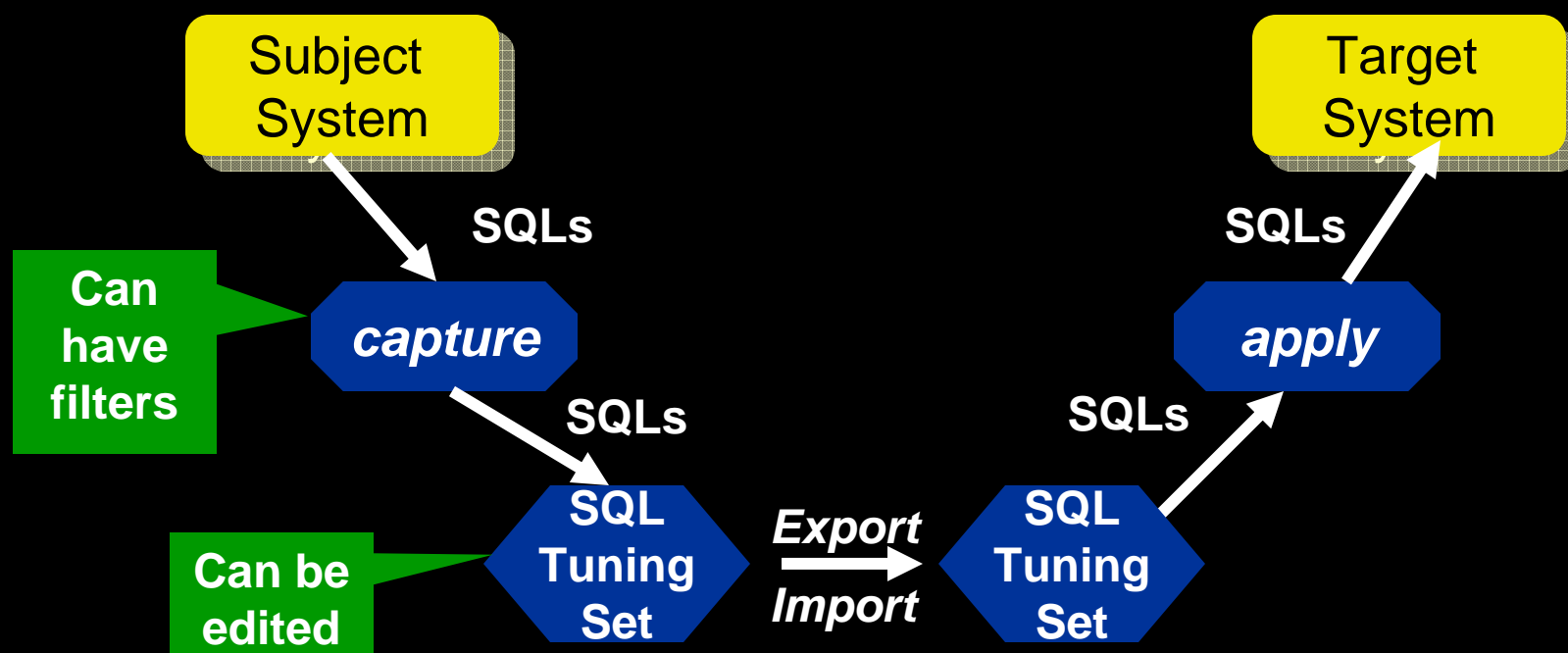
Compared to QA Tools

- How does it compare to QA tools like Load Runner?
  - QA tools use synthetic workload, i.e. the SQLs you provide to it. DBR uses the real SQLs that ran – good, bad and ugly
  - That's why it's called Real Application Testing (RAT)
  - QA Tools measure end to end app – webserver to app server to DB. DBR only measures the DB performance
- So, it's not a testing tool for your apps

Caveats

- DBR captures only the SQLs executed in the database; not the activity on the apps such as clicks.

- No guarantee of elapsed time between SQLs

- Concurrency of statements not guaranteed

SQL Performance Analyzer

- The other constituent of the RAT family
- Replays SQLs captured in SQL Tuning Sets

Subject System

Target System

SQLs

SQLs

Can have filters

capture

apply

SQLs

SQLs

Can be edited

SQL Tuning Set

Export

Import

SQL Tuning Set

Different from DR

- RAT – Real Application Testing
- DR captures all the SQLs.
  - You can apply filters; but not very flexible
- SPA allows powerful filters during capture
- It shows the SQLs
  - so you can remove them. DR can't show SQLs
- DR follows the sequence and repetition of SQLs; SPA does not.
- SPA is good for individual SQL tuning; DR is for DB.

Good for

- SPA is good for single SQL or single app
- Where concurrency is not important
- Checking if these are better:
  - Profiles
  - Outlines
  - Parameters – session/system
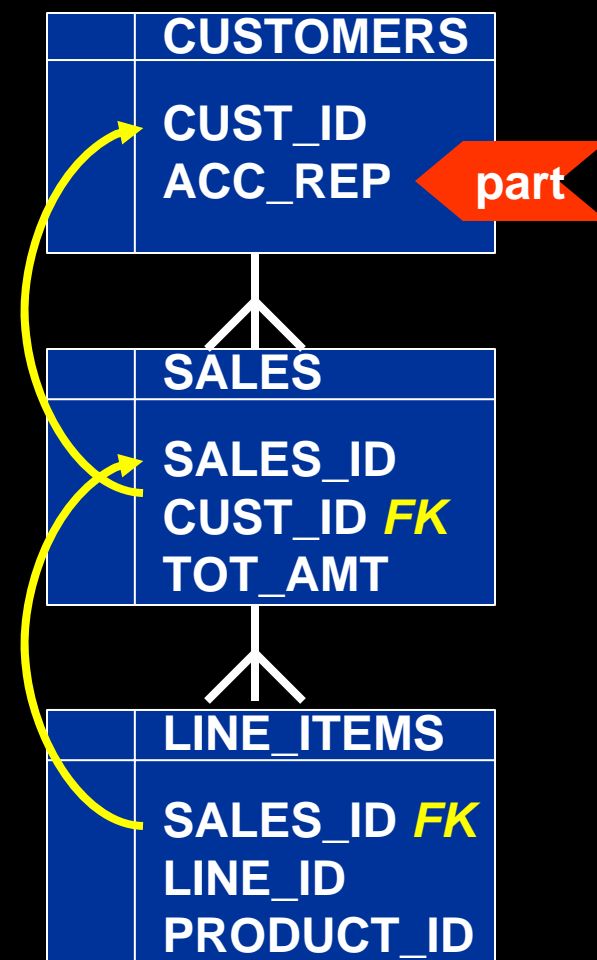
Upgrade from 10g to 11g

- The 10.2.0.4 patchkit has the capability to capture the SQLs to a Tuning Set

- It can't replay; only capture.

- MetaLink Note 560977.1 has one-off patches for pre-10g databases

Expanded Sub-Partitioning

- **New composite partitioning schemes**
  - Range-range
    - 2 date columns
  - Hash-range
    - PK first and then date
  - Hash-hash
    - PK and then another unique key
  - Hash-list
    - PK and discrete values
  - List-range

Referential Partitioning

- You want to partition CUSTOMERS on ACC_REP column

- The column is not present on child tables

- Earlier option: add the column to all tables and update it
  - Difficult and error-prone

- 11g has referential partitioning

CUSTOMERS

CUST_ID
ACC_REP ◀ **part**

SALES

SALES_ID
CUST_ID *FK*
TOT_AMT

LINE_ITEMS
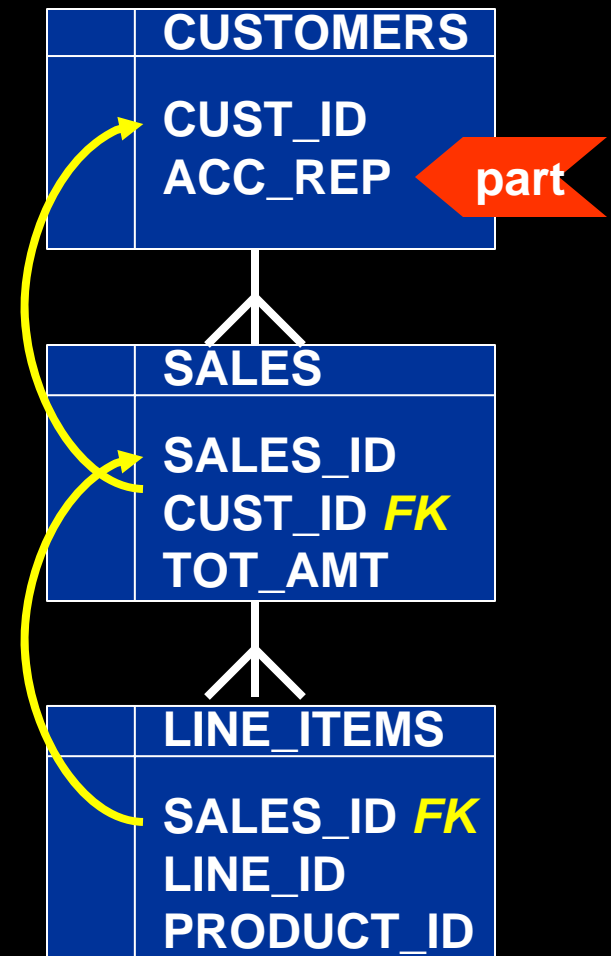
SALES_ID *FK*
LINE_ID
PRODUCT_ID

17

Referential Partitioning

Partition CUSTOMERS as usual

```
create table SALES (
   SALES_ID number not null,
   CUST_ID  number not null,
   TOT_AMT  number
   constraint fk_sales_01
      foreign key (cust_id)
      references customers)
partition by reference
   (fk_sales_01);
```

Partitions of SALES are created with
 data from CUSTOMERS.

CUSTOMERS

CUST_ID
ACC_REP   `part`

SALES

SALES_ID
CUST_ID *FK*
TOT_AMT

LINE_ITEMS

SALES_ID *FK*
LINE_ID
PRODUCT_ID

Addressing Ref Partitions

- USER_PART_TABLES view has info
  - partitioning_type – "REFERENCE"
  - ref_ptn_constraint_name – the FK name
- To address a specific partition (remember: you don't have a part name):
  - select * from sales partition for (to_date('15-may-2007','dd-mon-yyyy'));

INTERVAL Partitioning

- SALES table partitioned on SALES_DT
  - Partitions defined until SEP 2008. Before Oct starts, you have to create the partition
  - If you don't create the part, the INSERT will fail on Oct 1st.
- To mitigate the risk, you created the PMAX partition. *Undesirable*
- When you finally add the OCT08 partition, you will need to split the PMAX – *highly undesirable*

Interval Partitions

```
create table SALES ( sales_id
  number,
  sales_dt date )
partition by range (sales_dt)
interval
  (numtoyminterval(1,'MONTH'))
  store in (TS1,TS2,TS3)
( partition SEP08 values less than
  (to_date('2008-10-01', 'yyyy-mm-
  dd'))
);
```

Creates a partition automatically when a new row comes in

Specifies one partition per month

This is the first partition. The subsequent partition names are system generated

Addressing Interval Partitions

- USER_PART_TABLES view:
  - partitioning_type – "INTERVAL"
- USER_TAB_PARTITIONS view:
  - high_value shows the upper bound of partition
- To address a specific partition:

```
select * from SALES partition for
  (to_date('22-sep-2008','dd-mon-
  yyyy'));
```

Physical Standby

- Physical Standby Database with Real Time Apply

- Almost real time, savings in CPU, etc.

- But opening in read only access makes it miss the SLA

- So, the investment just sits idle => inefficient

1. Backups can be off this, less CPU load on primary
2. Can be open for Read Only access. Good for reporting
3. But if open, the recovery stops, defeating the purpose of standby

Primary   **Standby**

Standby Redo Logs
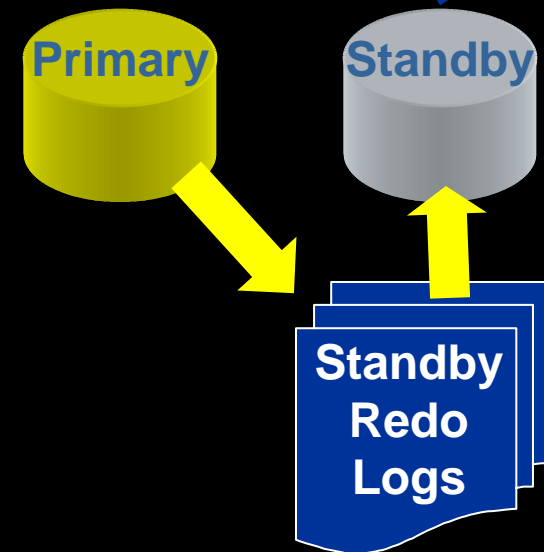
Active Data Guard

- Physical Standby Database with Real Time Apply
- But you can open the database in read only
- And then start the managed recovery process
- So, you meet the SLA for uptime while making efficient use of the investment.

1. Backups can be off this, less CPU load on primary
2. Can be open for Read Only access. Good for

Primary

Standby

Standby Redo Logs

Extra-cost option

24
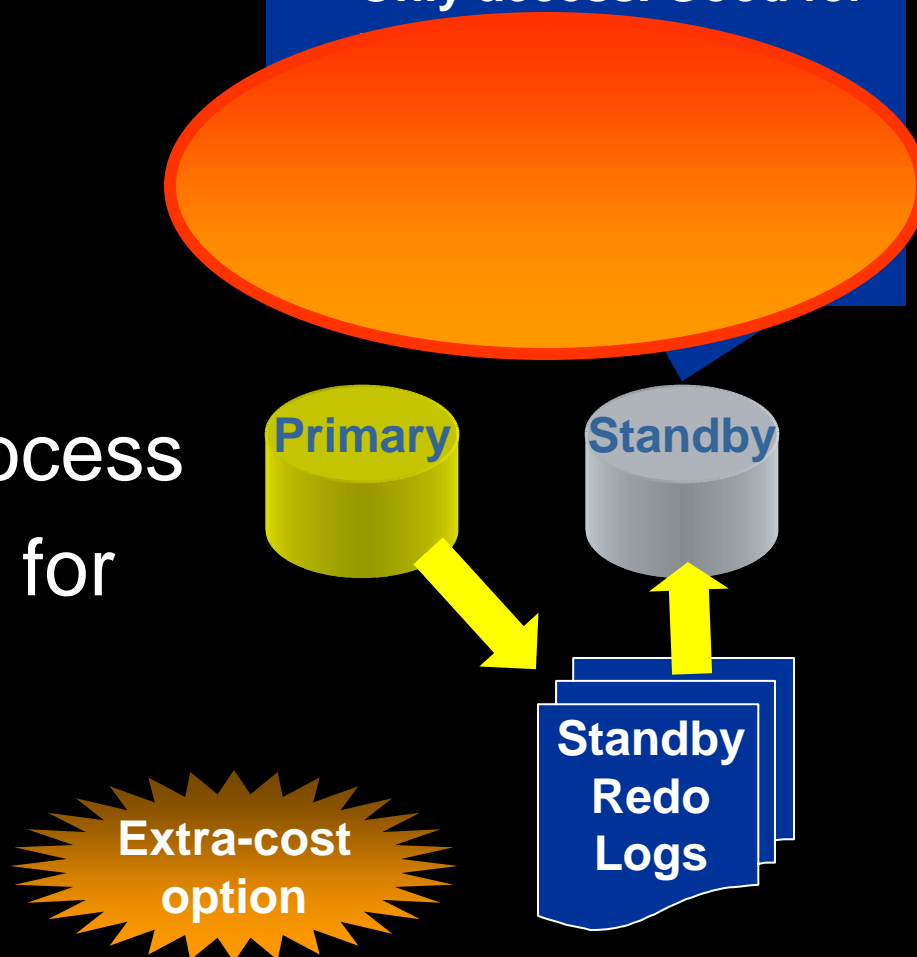
Comparison

10g	11g
Standby in managed recovery mode	Standby in managed recovery mode
alter database managed standby database cancel	alter database managed standby database cancel
alter database open read only	alter database open read only
shutdown, startup mount	alter database recover managed standby database disconnect
alter database recover managed standby database disconnect	

Snapshot Standby

- You can open a standby as read write

  ```
  alter database recover managed
    standby database cancel;
  alter database convert to snapshot
    standby;
  ```

- Do your testing

- Convert back to normal

  ```
  alter database convert to physical
    standby;
  ```

Other Enhancements

- Easier Creation
- Physical -> Logical; Back to Physical

```
alter database recover to logical
    standby DBName;

alter database start logical standby
    apply immediate;
```

- Archive Log Compression

```
alter system set log_archive_dest_2 =
    'service=pro11sb LGWR ASYNC
    valid_for=(online_logfiles,primary_role)
    db_unique_name=sby compression=enable'
```

Rolling Upgrades

1. Convert S to Logical
2. Reverse the roles P=standby, S=prima~~ry~~
3. Apps will move to S
4. Stop standby
5. Upgrade P
6. Reverse roles. P=primary, S=standby
7. Upgrade S
8. Convert back to Physical

primary
P
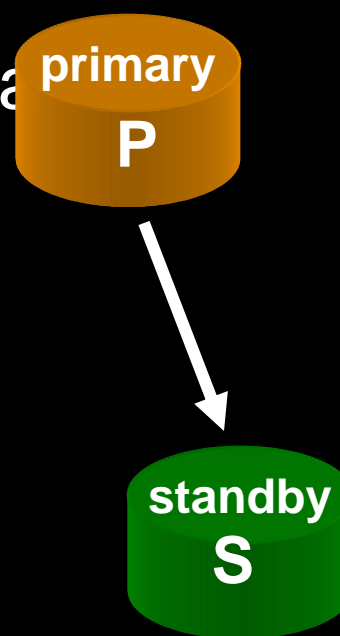
standby
S

Parameter Testing

1. Capture workload from P using Database Replay
2. Convert S to Snapshot Standby
3. Create a restore point rp1
4. Change parameter
5. Replay captured workload on S
6. Measure performance
7. Repeat with new values
8. Convert S back to physical

primary
P

standby
S
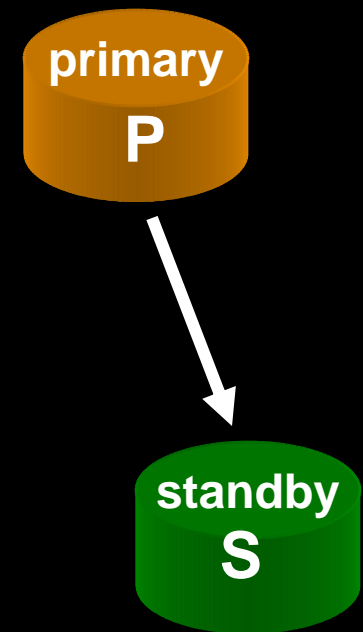
Plan Wreaks Havoc

- A Typical Scenario:
 - A specific SQL had a good plan
  - The stats on the table was collected
  - The plan changed for worse.
  - You get blamed for *collecting* stats. You stop collecting stats
- Another Scenario:
  - The same SQL had a good plan
  - Suddenly you saw performance issues
  - The cause was identified as stale stats
  - You re-collect stats
  - SQL performs well again
  - You get blamed for *not* collecting stats!

Typical Solutions

- Stored Outlines
  - Forces a plan
  - May be a bad plan later
- SQL Profiles
  - Data based; may be worse later
- Hints
  - Forces a plan which could be worse later
  - Not possible in canned apps
- Best pie-in-the-sky Solution
  - Examine the new plan; implement only if better

11g Plan Management

- If enabled, Oracle stores the SQL and the plan in a repository called SQL Management Base (SMB)
- When a new plan is generated, it is compared against the old plan
- If better, the new plan is implemented
- Else, the old plan is forced (like outlines)
- The DBA can examine the plans and force a specific plan

Query

optimization

plan

stored and compared

old plan

SMB
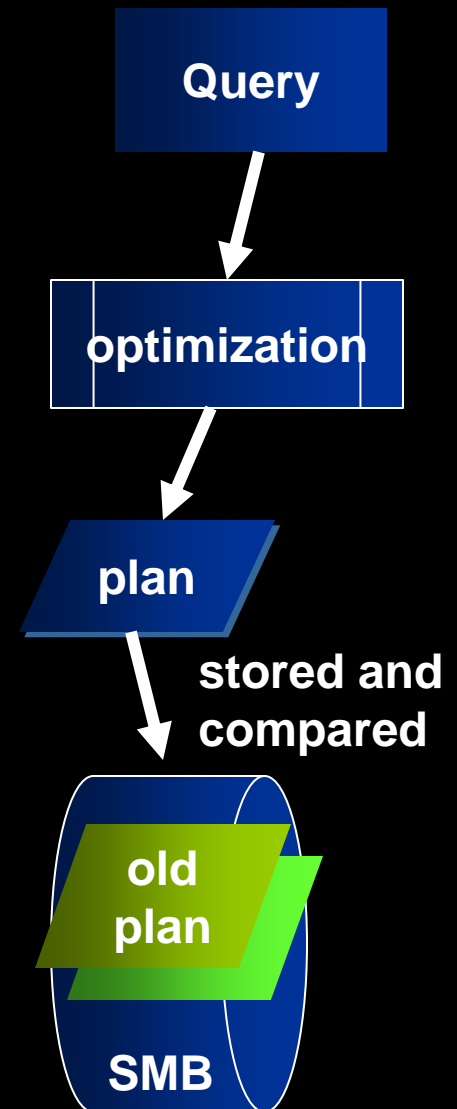
SQL Baselines

- Similar to Stored Outlines

```
SQL> alter system
  optimizer_capture_sql_plan_baselin
  es = true;
```

- All the baselines are captured

- Don't confuse with AWR Baselines

SQL Profile   **SQL Patch**   **SQL Plan Baseline**

Refresh

A SQL Plan Baseline is an execution plan deemed to have acceptable performance for a given SQL statement

Settings

Capture SQL Plan Baselines TRUE
Use SQL Plan Baselines TRUE
Plan Retention(Weeks) 53 Configure

Search

SQL Text % customers % > 3   Go

By default, the search returns all uppercase matches beginning with the string you entered. To make the search case-sensitive, enclose it in double quotes. To widen the search, add a percent sign before and/or after the search string. You can use the wildcard symbol (%) in a double quoted string.

Load   Unpack

Enable   Disable   Drop   Evolve   Pack   Fixed - Yes ▾ Go

Select All | Select None

Select	Name ▽	SQL Text	Enabled	Accepted	Fixed	Auto Purge	Created	Last Modified
☐	SYS_SQL_PLAN_b5429522ee05ab0e	select count(1) from customers where state_code = ...	YES	NO	NO	YES	Aug 15, 2007 5:04:22 PM	Aug 15, 2007 5:04:22 PM
☐	SYS_SQL_PLAN_b5429522e53beeec	select count(1) from customers where state_code = ...	YES	YES	NO	YES	Aug 13, 2007 12:25:29 AM	Aug 13, 2007 1:07:55 AM

TIP The table will display maximum of 2000 rows. Use search criteria to get the desired results.

- **Enabled – will it be considered or not?**
- **Accepted – Current plan by optimizer**
- **Fixed – the plan is fixed, i.e. optimizer forces it. Similar to outlines**
- **Auto Purged – after some days the plan is purged, unless accepted**

34

This is the SQL Plan Evolve Report

```
Inputs:
-------
  PLAN_LIST  = SYS_SQL_PLAN_b5429522ee05ab0e
               SYS_SQL_PLAN_b5429522e53beeec
  TIME_LIMIT = DBMS_SPM.AUTO_LIMIT
  VERIFY     = YES
  COMMIT     = YES


Plan: SYS_SQL_PLAN_b5429522e53beeec
-------------------------------------
  It is already an accepted plan.


Plan: SYS_SQL_PLAN_b5429522ee05ab0e
-------------------------------------
  Plan was verified: Time used 3.9 seconds.
  Failed performance criterion: Compound improvement ratio <= 1.4.


                          Baseline Plan      Test Plan      Improv. Ratio
                          -------------      ---------      -------------
  Execution Status:            COMPLETE       COMPLETE
  Rows Processed:                     1              1
  Elapsed Time(ms):                3396            440               7.72
  CPU Time(ms):                    1990            408               4.88
  Buffer Gets:                     7048           5140               1.37
  Disk Reads:                      4732             53              89.28
  Direct Writes:                      0              0
  Fetches:                         4732             25             189.28
  Executions:                         1              1
```

You can examine the baselined plan and the newly calculated plan. If the new one looks better, you can force it, called "Evolve".

Testing Statistics

- ## Scenario
  - SQL was performing well
  - You want to collect stats
  - But you hesitate … will be make it worse?

- ## How do you make sure?
  - Collect the stats and run the SQL
  - Are you kidding … in prod?!!!!

Private Statistics

1. Mark a table's stats as private
2. Collect stats; but optimizer will not see
3. Issue `alter session set optimizer_use_pending_statistics = true;`
4. Now optimizer will see the new stats in that session alone
5. Test SQL. If OK, publish stats:
   `dbms_stats.publish_pending_stats('`*Schema*`', '`*TableName*`');`

Further Notes

- You set a table's preference:
  ```
  dbms_stats.set_table_prefs (
  ownname => 'Schema',
  tabname => 'TableName',
  pname   => 'PUBLISH',
  pvalue  => 'FALSE' );
  ```

- Now the table's stats will always be private until you publish them

- You can delete private stats:
  ```
  dbms_stats.delete_pending_stats
  ('Schema','Table');
  ```

Stats History

- ## History
  ```
  desc DBA_TAB_STATS_HISTORY
      OWNER
      TABLE_NAME
      PARTITION_NAME
      SUBPARTITION_NAME
      STATS_UPDATE_TIME
  ```

- ## Reinstate previously gathered stats
  ```
  dbms_stats.restore_table_stats (
  ownname => 'Schema',
  tabname => 'TableName',
  as_of_timestamp => '14-SEP-07 11:59:00
    AM' );
  ```

Encrypted Tablespaces

- Transparent Data Encryption (TDE) allows column level encryption

- Performance hit, since index scans can't be used and every time the data has to be decrypted to be compared

```
select *
from payments
where CC#
like '1234%'
```
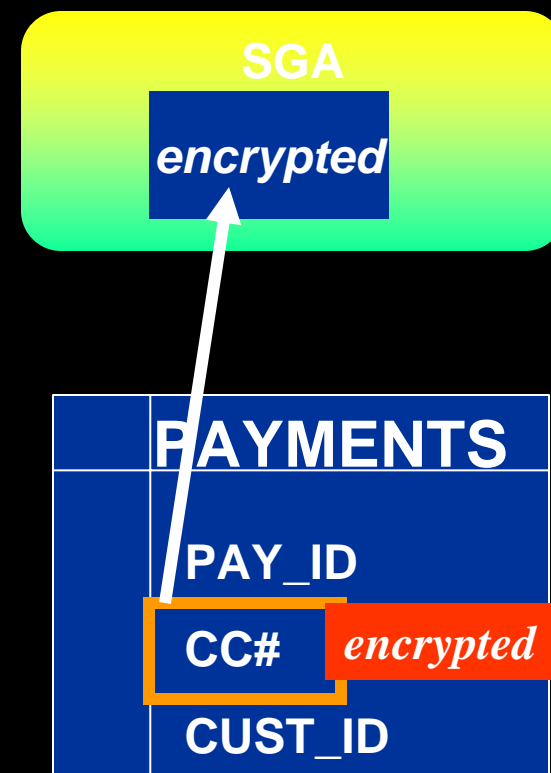
Transparent Tablespace Encryption

- Entire tablespace is encrypted

```
create tablespace secure1
datafile '/db1/1.dbf' size 1M
encryption using 'AES128'
default storage (encrypt)
```

- All objects stored in the tablespace are encrypted, all columns

- But when they are loaded to the SGA, they are in cleartext

- So index scans are a good

```
select *
from payments
where CC#
like '1234%'
```

SGA

clear

PAYMENTS

PAY_ID

CC#   *encrypted*

CUST_ID
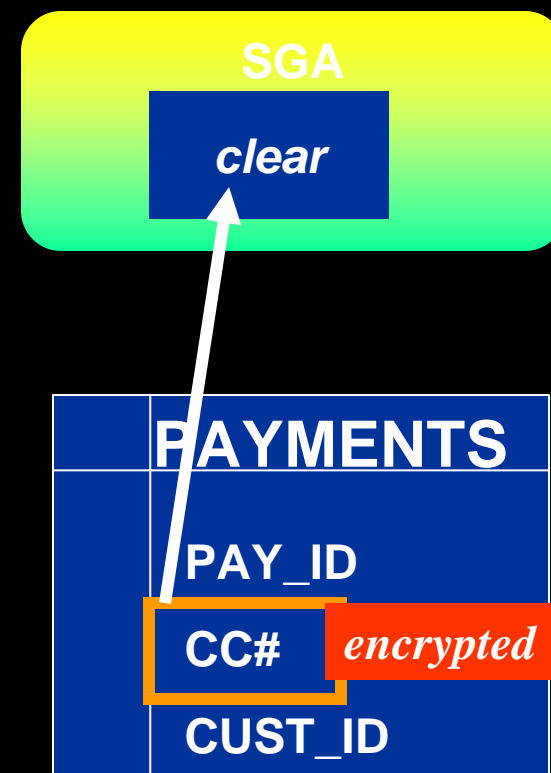
Dictionary

```
SQL> desc v$encrypted_tablespaces
 Name                        Null?      Type
 ---------------- -------- -----------
 TS#                                     NUMBER
 ENCRYPTIONALG                           VARCHAR2(7)
 ENCRYPTEDTS                             VARCHAR2(3)
```

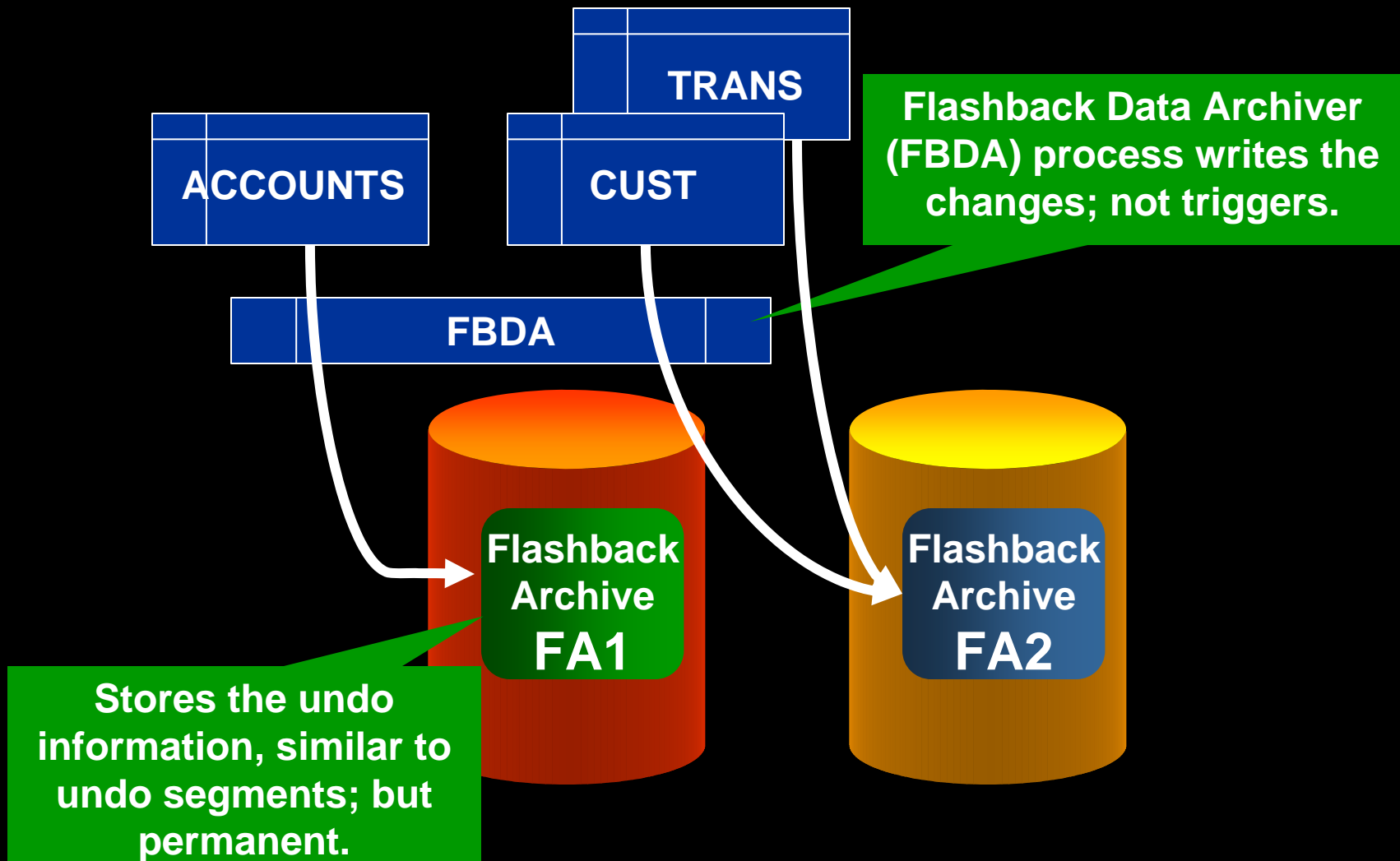- The column ENCRYPT_IN_BACKUP in V$TABLESPACE shows the encryption during RMAN backup

Data as of Previous Time

- Flashback Queries (9i)

```
select * from accounts
as of timestamp to_date
  ('09/18/2008','mm/dd/yyyy');
where acc_no = 1801;
```

- Gets information from Undo Segments

- When undo gets filled up, the information is gone. Not reliable.

- *Solution* – triggers to populate user defined change tables.

Flashback Data Archives

TRANS

ACCOUNTS

CUST

Flashback Data Archiver (FBDA) process writes the changes; not triggers.

FBDA

Flashback
Archive
FA1

Flashback
Archive
FA2

Stores the undo information, similar to undo segments; but permanent.

Syntax

- ## Create a FB Archive
  ```
  create flashback archive FB1
  tablespace TS1
  retention 1 year
  ```

- ## Attach FBA to a table:
  ```
  alter table ACCOUNTS flashback
    archive FA1;
  ```

- ## Purges automatically. Manually:
  ```
  alter flashback archive FA1 purge
    before scn 1234567;
  ```

Comparison w/Triggers

- Manually create change tables and trigger logic
- The triggers can be disabled, making it legally non-binding
- Change tables can be deleted by DBA, so not immutable.
- Triggers do a context switch; FBAR process runs in the background with minimal impact.
- Purging is not automatic

Usage

- Just normal flashback query:

  ```
  select … from accounts as of …
  ```

- Purge is automatic after the retention period. Manually possible too.

- DBA can't modify data; so legally binding.

PL/SQL Native Compilation

- PL/SQL can be compiled two ways:
  - *Interpreted*, resulting in m-code, which only the PL/SQL compiler can interpret
  - *Native*, which creates a C-code from PL/SQL, which is then stored as an O/S resident library
- Faster for non-data portions
- Requires C library
- Usually not available in production systems

11g Way

```
SQL> alter session set
  plsql_code_type = native;
SQL> alter procedure p1 compile;
```

- C-complier is built into the database
- Compilation Time(plsql_optimize_level=2)

	10g	11g
Interpreted	1.66	1.64
Native	4.66	2.81

- Computation intensive code will benefit. Data manipulation code will not.

Caching

- Query is often executed on tables that do not change much.

- Typical Solution: Materialized Views
  - Results are already available; no need to re-execute the query
  - Results could be stale; not updated unless refreshed
  - Underlying data doesn't change; but MV doesn't know that, unless fast refresh

- Not practical

Result Cache

- `select /*+ result_cache */ …`
- The results of the query are stored in the SGA
- Result Cache – a new area in SGA
- `result_cache_max_size` states the size of RC
- The query executes as usual if the cache is not found
- The cache is refreshed automatically when the underlying data changes

DDL Waits

- Session 1:

  ```
  update t1 set col1 = 2;
  ```

- Session 2:

  ```
  alter table t1 drop column col2
                *
  ERROR at line 1:
  ORA-00054: resource busy and acquire with NOWAIT
     specified or timeout expired
  ```

- In a busy system you will never get the exclusive lock.

- In 11g

  ```
  alter session set ddl_lock_timeout = 15;
  ```

- This will make the session wait for 15 seconds before erroring with ORA-54.

Trigger Execution

- You have 3 pre-insert triggers tr1, tr2 and tr3.

- How do you make sure they fire in that sequence?

- You can, now in 11g.
  ```
  create trigger tr3
  before insert on TableName
  follows tr2
  begin
  ...
  ```

Case Sensitive Passwords

- 11g compliant password allows you differentiate between "tiger" and "TIGER"
- Init Parameter
  `sec_case_sensitive_logon = true`
  enables it
- Dynamic – `ALTER SYSTEM SET` …

Upgrade Advice

1. Use snapshot standby to test your upgrade process
2. Use Workload Capture in 10g and replay in snapshot standby
3. Modify parameters, replay and modify: repeat until you get it right
4. Use SQL Performance Analyzer to test the handful of errant queries
5. Use SQL Baselines to fix them

- All the concepts are described in detail in the OTN series:

    http://www.oracle.com/technology/pub/articles/oracle-database-11g-top-features/index.html

ORACLE DATABASE **11**^{g} Oracle Database 11g:
The Top New Features for DBAs and Developers
by Arup Nanda

Thank You!