



Oracle9i New Features for Developers

Presented at NYOUG
March 13, 2003
Dave Anderson
Dave@skillbuilders.com

SKILLBUILDERS



0.2

Goal

- A brief introduction to 9i features and enhancements impacting developers
- Will be moving fast – lots to cover
- See some examples
- Grasp understanding of purpose of feature or enhancement

© 2003 SkillBuilders, Inc.



0.3

Topics

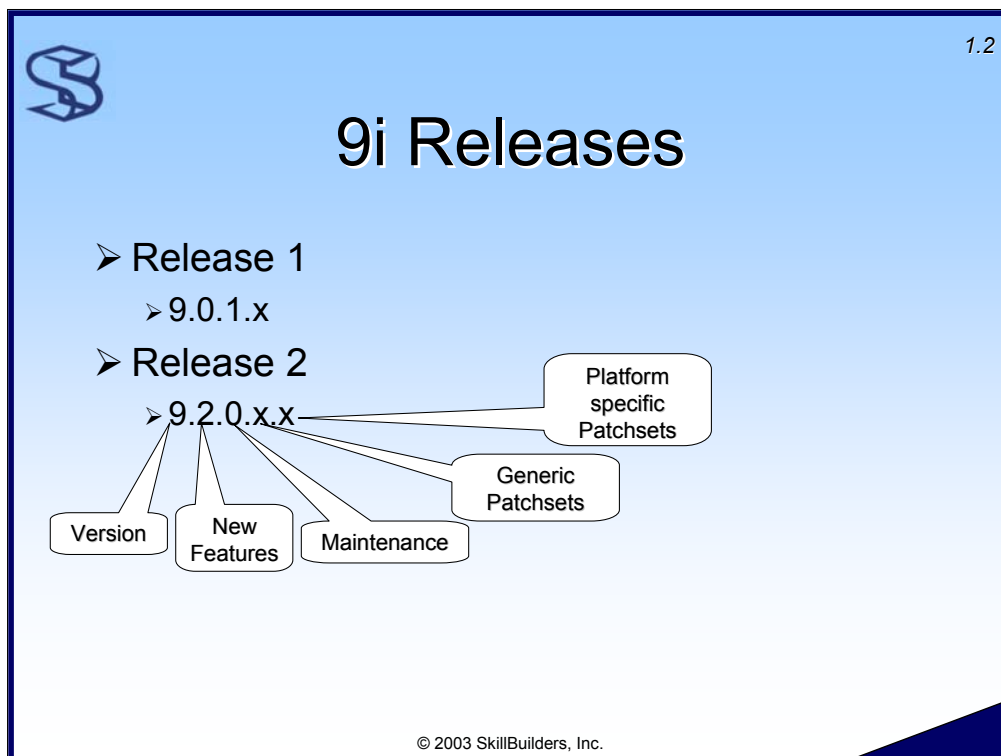
- Deprecated Features
- Flashback Query
- External Tables
- SQL Features
- PL/SQL Features
- New Datatypes
- Resumable Space Mgmt
- Tuning Enhancements
- Security
- Index Enhancements

© 2003 SkillBuilders, Inc.

1. Deprecated Features for Developers

A brief look at what deprecated
features will impact developers.

SKILLBUILDERS



1.3

Deprecated Features

- **Deprecated**
 - Still supported but not recommended for use
 - Planned for desupport in a future release
- **ANALYZE** command to collect statistics
- **Export / Import** `INCREMENTAL` functionality
- `LONG`, `LONG RAW` data types
- Let's look at each in turn...

© 2003 SkillBuilders, Inc.

Deprecated – Still supported but not recommended for use. If a feature has been listed as deprecated you should begin to plan for its desupport in a future release. This gives you time to plan for this upcoming change. You should have some idea of it's impact.


Desupported – No longer supported in a release. These features just will not work. In some cases there is an alternative way to accomplish this feature, in other cases, this type of feature is just no longer available within this product.

These developer-related features are deprecated in Oracle9i:

- **Export / Import** `INCREMENTAL` functionality
- `LONG`, `LONG RAW` data types
- **ANALYZE** command to collect statistics

There are also some DBA-related features that have been deprecated:

- Some `init.ora` parameters
- `bstat` / `estat` scripts (replaced by `STATSPACK`)



LONG, LONG RAW Data Types


1.4

- **Deprecated**
 - **LONG** – Variable length character datatype
 - Use CLOB instead
 - **LONG RAW** – Variable length raw (binary)
 - Use BLOB or BFILE instead
 - **R2 enhanced support for LOBs**
 - All SQL functions supported
 - Comparison = < > supported
 - No code changes necessary to implement LOBs

© 2003 SkillBuilders, Inc.

The `LONG` and `LONG RAW` datatypes are provided for backward compatibility and have been deprecated.

LOB datatypes such as `CLOB`, `BLOB`, and `BFILE` should be used instead of `LONG` and `LONG RAW`.

1.5

ANALYZE Command

➤ Deprecated, use DBMS_STATS package instead

```
SQL> begin
2  dbms_stats.gather_table_stats(user , 'employee',
3      cascade=>TRUE,
4      method_opt=> 'FOR ALL INDEXED COLUMNS');
5  end;
6  /
```

PL/SQL procedure successfully completed.

➤ More flexible - several procedures available

- GATHER_INDEX_STATS, GATHER_TABLE_STATS
- GATHER_SCHEMA_STATS, GATHER_DATABASE_STATS

© 2003 SkillBuilders, Inc.


The use of the `ANALYZE` command to gather statistics has been deprecated and may be desupported in future releases.

`DBMS_STATS` is more flexible in that there are several procedures that providing different target scope, i.e. you can collect statistics on an individual object (table or index), all objects within a schema or even the entire database.

In this example I use the `GATHER_TABLE_STATS` procedure to collect statistics for my `EMPLOYEE` table and all dependent indexes. Unlike the `ANALYZE` command (deprecated), the default is *not* to collect stats on dependent indexes, so you must use `CASCADE =>TRUE` parameter.

The `METHOD_OPT` parameter controls the creation of histograms. (See the section on histograms later in this module.) Histograms are expensive, so only create them on indexed columns.

Warning: The default is to create histograms on every column. This is rarely useful and very expensive. Use the `method_opt=> 'FOR ALL INDEXED COLUMNS'` clause or variant, which will create histograms only for some columns.

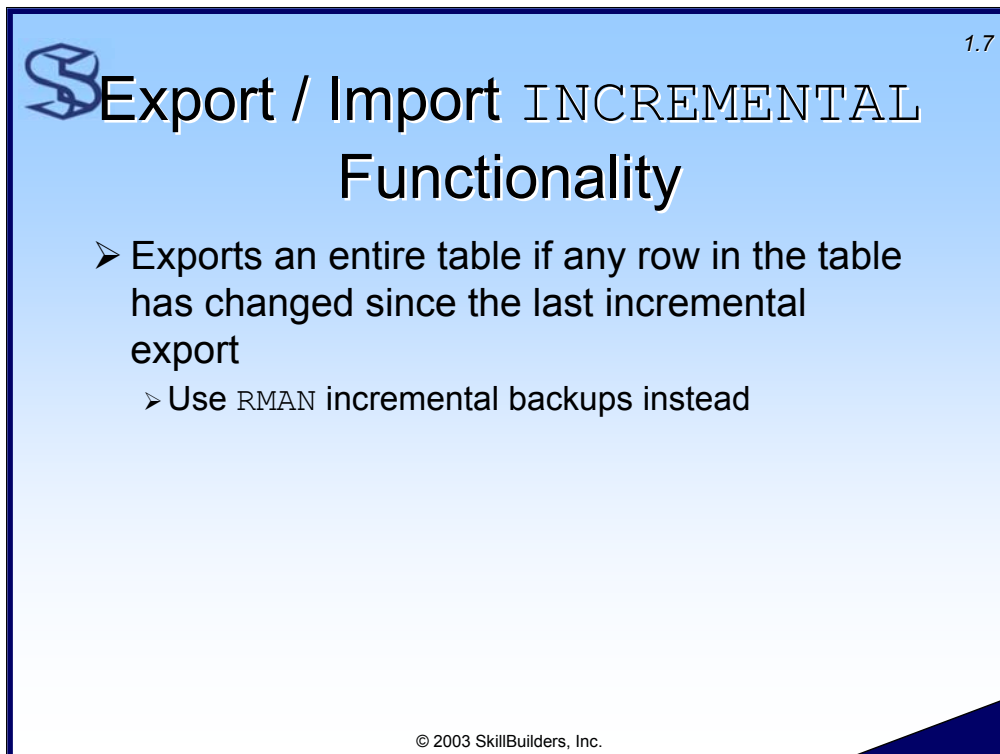
1.6

ANALYZE Command (cont)

- **DBMS_STATS advantages**
 - Can be parallelized for better performance
 - See DEGREE argument
 - Provides more accurate stats for partitioned objects
 - Backup stats in user tables
 - Prevent cursor invalidation with NO_INVALIDATE
 - Fabricate stats with SET to see effect
 - Export table stats, import into TEMP table
 - Can't gather for TEMP tables

© 2003 SkillBuilders, Inc.

The `DBMS_STATS` package should be used instead of the `ANALYZE` command to gather statistics. `DBMS_STATS` can be run in parallel, providing better performance. Also, Oracle documentation says that `DBMS_STATS` provides “more accurate” statistics than `ANALYZE` for partitioned objects.



1.7

Export / Import INCREMENTAL Functionality


- Exports an entire table if any row in the table has changed since the last incremental export
 - Use RMAN incremental backups instead

© 2003 SkillBuilders, Inc.

The INCREMENTAL functionality of the export / import utility has been deprecated.

Tables are still be able to be exported.

Instead, use RMAN incremental backups to back up your data. Remember, if you want to isolate a table for backup, you can always place a table in it's own tablespace.

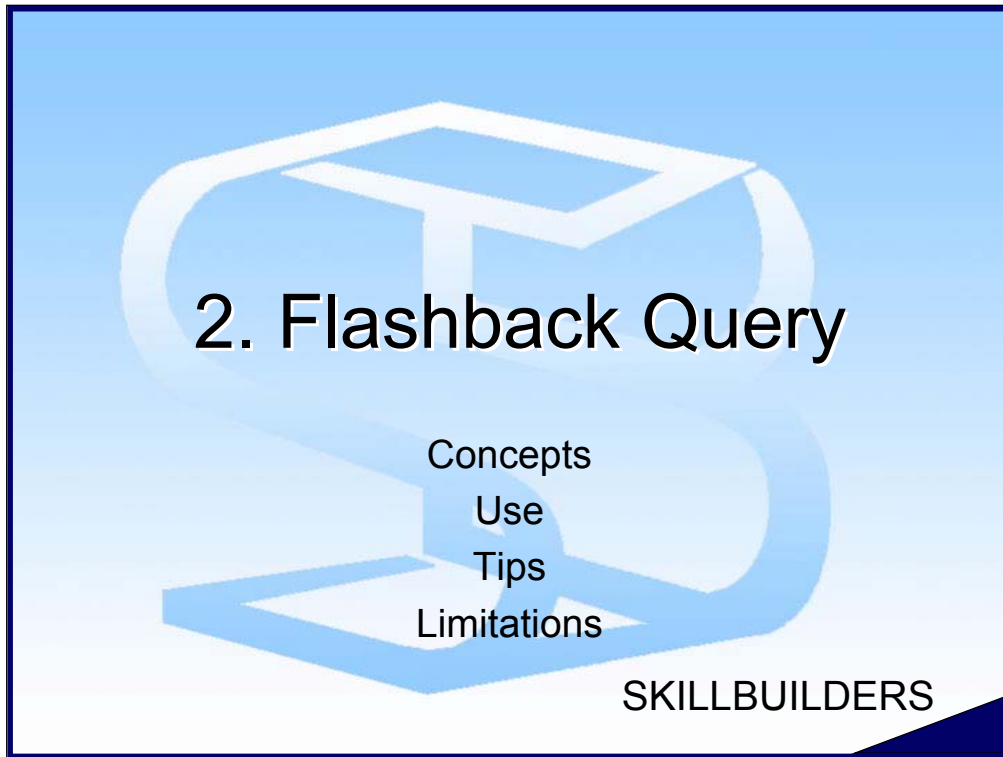
1.8


Summary

- **Deprecated Features**
 - `ANALYZE` command
 - Export / Import `INCREMENTAL` functionality
 - `LONG`, `LONG RAW` data types
- **Also note: Desupported Features**
 - Server Manager
 - `CONNECT INTERNAL`
 - Various `init.ora` parameters
 - Changes to data dictionary
 - E.g. no more `DBA_SNAPSHOT_LOGS`

© 2003 SkillBuilders, Inc.

Be aware of what is coming up in future releases of Oracle. Prior to new releases many commands, parameters, views, utilities, and datatypes may have been deprecated, which means they are still supported but may become desupported or obsolete in a future release.



2.2

Concepts


- Execute `SELECT` as if it was being run in the past
 - Database restore not necessary
 - “Self-service error correction”
- Window depends on `UNDO_RETENTION` parameter
 - DBA sets in `INIT.ORA` or `SPFILE`
 - For example, retain 12 hours of UNDO
 - `undo_retention = 43200`
- Should use 9i automatic UNDO management
 - `undo_management = AUTO`

© 2003 SkillBuilders, Inc.

Flashback Queries are a new type of query that allows one to get the results of a query as if one was running it in the past (a previous point in time). Up to now one would need a database restore to accomplish the same. With Flashback Queries one can achieve this without involving the DBA.

This feature is particularly valuable if one accidentally deletes rows from a table. Using Flashback one can easily recover the deleted rows and (perhaps) save them to a temporary file/table. This is where the Oracle marketing phrase “self-service error correction” came from.

How far back into the past one can go is dependent on the value of the parameter `UNDO_RETENTION`. This is usually specified by the DBA in `INIT.ORA` or `SPFILE`. Note that the database should be configured to use ‘Automatic Undo Management’ rather than traditional rollback segments so that the retention window can be specified (`undo_retention` parameter).

2.3

DBMS_FLASHBACK

- DBMS_FLASHBACK package manages flashback mode
- Set flashback mode to approximately 1 hour ago:

```
exec DBMS_FLASHBACK.ENABLE_AT_TIME (SYSDATE - (1/24));  
SELECT * FROM ord;  
exec DBMS_FLASHBACK.DISABLE;
```
- Scope is session
 - Database-wide scope not available
- Need execute privilege on dbms_flashback

© 2003 SkillBuilders, Inc.

The `dbms_flashback` package is one method used to enable flashback mode. The `ENABLE_AT_TIME` procedure enables flashback mode to the specified time. For flashback support, Oracle saves an SCN every five minutes. The time specified in the `ENABLE_AT_TIME` procedure is converted to the next lower SCN.

Only the current session is affected by flashback query mode. Database-wide support is not available. Users will need `EXECUTE` privilege on the `dbms_flashback` supplied package.


Supplemental Notes:

To change the flashback time, you must disable flashback mode and re-enable. See the `DISABLE` example later in this section.

Error “ORA-08180: no snapshot found based on specified time” if flashback time too old, i.e. an earlier time than any `UNDO` data exists for.

Error “ORA-01466: unable to read data - table definition has changed” returned if a table involved in the flashback operation (`SELECT`, `INSERT`) did not exist at the specified flashback time or was altered after the specified flashback time.

Error “ORA-08182: operation not supported while in Flashback mode” if DML is attempted while in flashback mode.

2.4

Statement Level Flashback

➤ 9i Release 2 provides statement level support

```
select lastname
from customer AS OF timestamp
to_timestamp('2003-01-08 05:30:00', 'YYYY-MM-DD HH:MI:SS')
```

```
select lastname
from employee
minus
select lastname
from employee as of timestamp
to_timestamp('2003-03-09 05:30:00',
'YYYY-MM-DD HH:MI:SS');
```

© 2003 SkillBuilders, Inc.

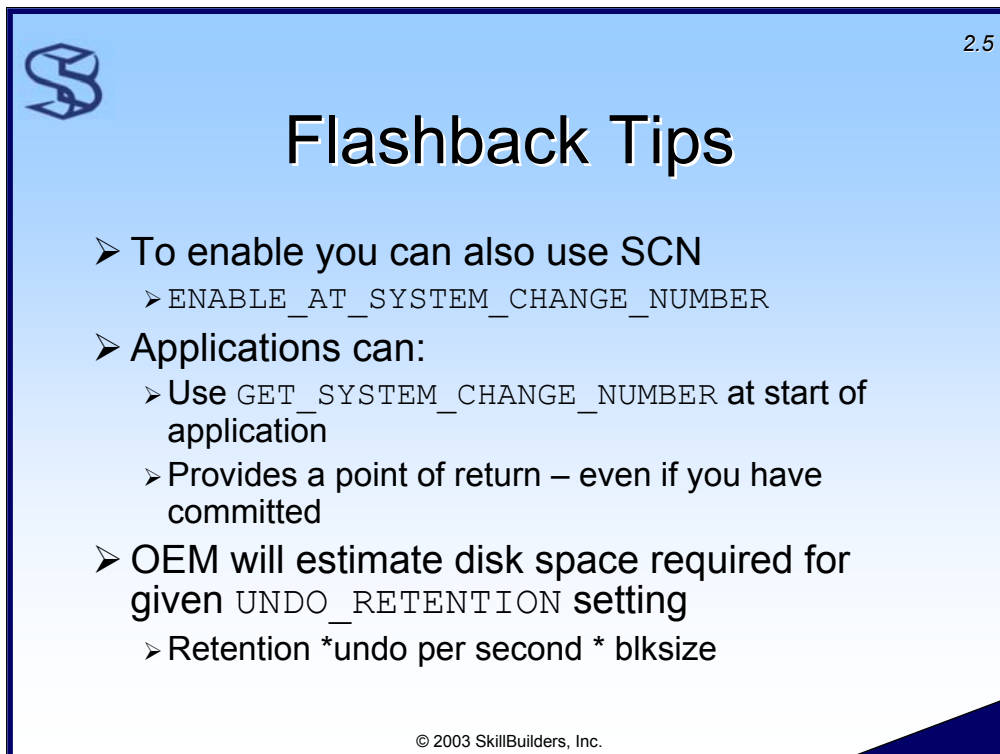
Oracle9i Release 2 supports statement-level flashback query. This is implemented through the “AS OF” clause of the `SELECT` statement.

Supplemental Notes

- 9i Release 2 provides robust support for statement-level flashback, including joins, subqueries, set operations and views using different time or SCN
- You can even create a view based on a flashback query. For example, I'd like to see my customer table as of a day ago:

```
create or replace view old_customer as
select lastname
from customer
AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' DAY);
```

- Note the use of the new `SYSTIMESTAMP` function and the `INTERVAL` datatype.
- It is also now easy to restore deleted rows (even if committed) by using flashback query within `INSERT ... SELECT` and `CREATE TABLE AS SELECT` commands

A presentation slide titled "Flashback Tips" with a blue background and a dark blue border. In the top left corner is a stylized 'S' logo. In the top right corner is the number "2.5". The slide contains a bulleted list of tips. At the bottom center, there is a small copyright notice: "© 2003 SkillBuilders, Inc.".

Flashback Tips

- To enable you can also use SCN
 - `ENABLE_AT_SYSTEM_CHANGE_NUMBER`
- Applications can:
 - Use `GET_SYSTEM_CHANGE_NUMBER` at start of application
 - Provides a point of return – even if you have committed
- OEM will estimate disk space required for given `UNDO_RETENTION` setting
 - $\text{Retention} * \text{undo per second} * \text{blksize}$

© 2003 SkillBuilders, Inc.

Example of enabling flashback mode with a SCN:

```
DECLARE
    old_scn NUMBER := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
BEGIN
    .
    .
    .
    EXECUTE DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER(old_scn);
END;
```

See Oracle9i Application Developer's Guide – Fundamentals for more info on use of Flashback Query. See Administration Guide for more info on setup of UNDO management.

Note that Oracle Enterprise Manager will estimate the disk space required for a given `UNDO_RETENTION` setting. A rough calculation can be made manually:

$\text{Retention setting} * \text{undo generated per second (see v\$undostat)} * \text{block size}$



2.6

Flashback Limitations

- DDL and DML not supported while in flashback mode
- Table structure changes invalidates old UNDO
 - E.g. DROP COLUMN
- Cannot flashback in middle of transaction (ORA-08183)
- ENABLE_AT_TIME maximum SYSDATE-5
 - Server uptime (not clock time)
 - Use SCN to go back farther
- Remote access via DBLINK not supported

© 2003 SkillBuilders, Inc.

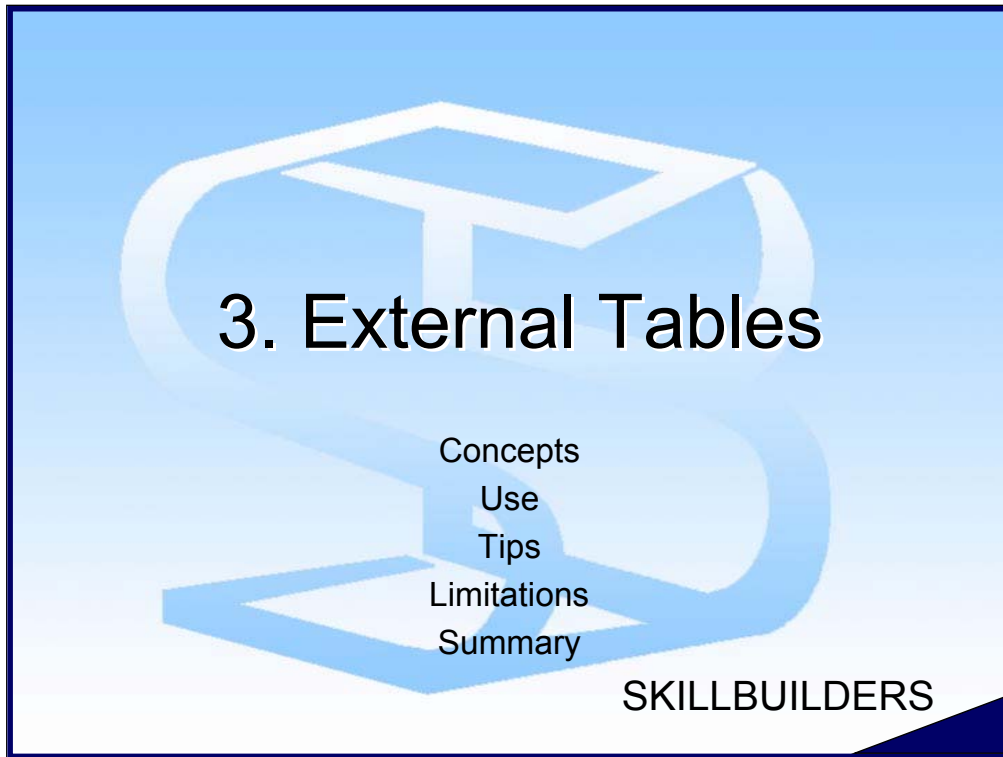



2.7

Summary

- Flashback Query allows `SELECT` as if it was being run at a previous point in time
- Provides “self-service error correction”
- Should use Automatic Undo Management
- Enable with `DBMS_FLASHBACK` package or “AS OF” clause on `SELECT`
 - Use Time or SCN

© 2003 SkillBuilders, Inc.





3.2

External Tables - Concepts


- Allows one to access external files as if it were a table
- Provides a good Extraction, Transformation and Load (ETL) tool
- Table can only be accessed in read-only mode
- All SQL query operations supported
 - Select, Join, MERGE, source for INSERT, multi-table INSERT, Views
- Needs an Access Driver
 - Program that can parse flat file
- ORACLE_LOADER Access Driver is supplied
 - It is really SQL*Loader

© 2003 SkillBuilders, Inc.

The new “`ORGANIZATION EXTERNAL`” clause of the `CREATE TABLE` command is used to access a flat file as if it were an Oracle table. The flat file data can be queried, but not updated. Indexes are not supported.

External files can be of any format as long as the “Access Driver” can perform the necessary conversions.

The Access Driver is the program responsible to read the external file into Oracle9i. Oracle provides a generic access driver called “`ORACLE_LOADER`”, which is really SQL*Loader.

3.3

External Tables – Use...


- First create Oracle DIRECTORY
- Directory contains OS file(s)
 - Contains data!

```
create or replace directory alert as  
  'c:\Oracle\admin\dave\bdump';  
  
create or replace directory oraclassdir as  
  'd:\oraclass\Labs';  
  
grant read on directory external_tables to  
  public;
```

© 2003 SkillBuilders, Inc.

The starting point for using external tables is to create an Oracle directory. The directory is an Oracle object that points to a server-based operating system directory. `CREATE ANY DIRECTORY` privilege is required to create a directory. The database will require OS privileges to read/write to the OS directory named in the `CREATE DIRECTORY` commands (typically, this is accomplished by granting the OS privileges to the OS user “ORACLE”).

Like all objects, the directory is protected from other database users. You must `GRANT READ` privilege on it to users who need query capability.

3.4

...External Tables – Use...

```
SQL> CREATE TABLE alert_log_ext
2      (detail_line VARCHAR2(2000))
3  ORGANIZATION EXTERNAL
4  (
5    TYPE oracle_loader
6    DEFAULT DIRECTORY alert
7    ACCESS PARAMETERS
8      ( RECORDS DELIMITED BY NEWLINE
9      nobadfile  nologfile  nodiscardfile
10     FIELDS      (detail_line char(80) ) )
11  LOCATION('alert_dave.log') )
12  REJECT LIMIT UNLIMITED;
```

Table created.

© 2003 SkillBuilders, Inc.

After the `DIRECTORY` has been created, we can create the external table. The example above illustrates creating an external table based on the database alert log.

We can see that the “`ACCESS PARAMETERS`” are just SQL*Loader parameters. See chapter 11 and 12 of the **Oracle9i Utilities** manual for more information about External Tables and Access Parameters.



3.5

...External Tables – Use...

```
select * from alert_log_ext  
where detail_line like '%ORA-%'  
/
```

Query
external table

```
create table alert_log  
as select * from alert_log_ext  
/
```

Load external table into a
permanent table


```
insert /*+ append */ into alert_log  
select * from alert_log_ext;
```

Load with direct
path INSERT

```
create table temp as  
select upper(detail_line) as detail_line  
from alert_log_ext  
/
```

Load with SQL
functions

© 2003 SkillBuilders, Inc.

3.6

...External Tables – Use

```
SQL> create table student_emails_ext
2  (firstname   varchar(40),
3  lastname    varchar(40),
4  email       varchar(80) )
5  organization external
6  (
7    type oracle_loader
8    default directory external_tables
9    location ('students_test.txt', 'students_test2.txt')
10 ) reject limit unlimited;
```

Table created.

```
SQL> select * from student_emails_ext;
```

| FIRSTNAME | LASTNAME | EMAIL |
|-----------|----------|-----------------------|
| A | Pan | apandaiah@hotmail.com |

© 2003 SkillBuilders, Inc.

This example uses all defaults for the `ACCESS PARAMETERS`.

We also can see the specification of multiple files in the `LOCATION` parameter.



3.7

Tips...

- Can use parallel load on fixed length files
- Generate external table DDL / Access Parameters from existing SQL*Loader control files

```
sqlldr scott/tiger ulcase1 EXTERNAL_TABLE=GENERATE_ONLY
```

- Gather statistics:

```
SQL> exec dbms_stats.gather_table_stats('system',  
    'student_emails_ext')
```

```
PL/SQL procedure successfully completed.
```

© 2003 SkillBuilders, Inc.



3.8

...Tips

- Combine with pipelined functions to create powerful ETL functions
- New dictionary views
 - *_EXTERNAL_TABLES
 - *_EXTERNAL_LOCATIONS
- Hints are said to work

© 2003 SkillBuilders, Inc.



3.9

Limitations

- Read only
- CONTINUEIF or CONCATENATE
 - Cannot combine multiple physical records into a single logical record
- Datatypes
 - GRAPHIC, GRAPHIC EXTERNAL, and VARGRAPHIC
 - CLOBs, NCLOBs, BLOBs, LONGs
 - nested tables, VARRAYs, REFs, primary key REFs, and SIDs

© 2003 SkillBuilders, Inc.

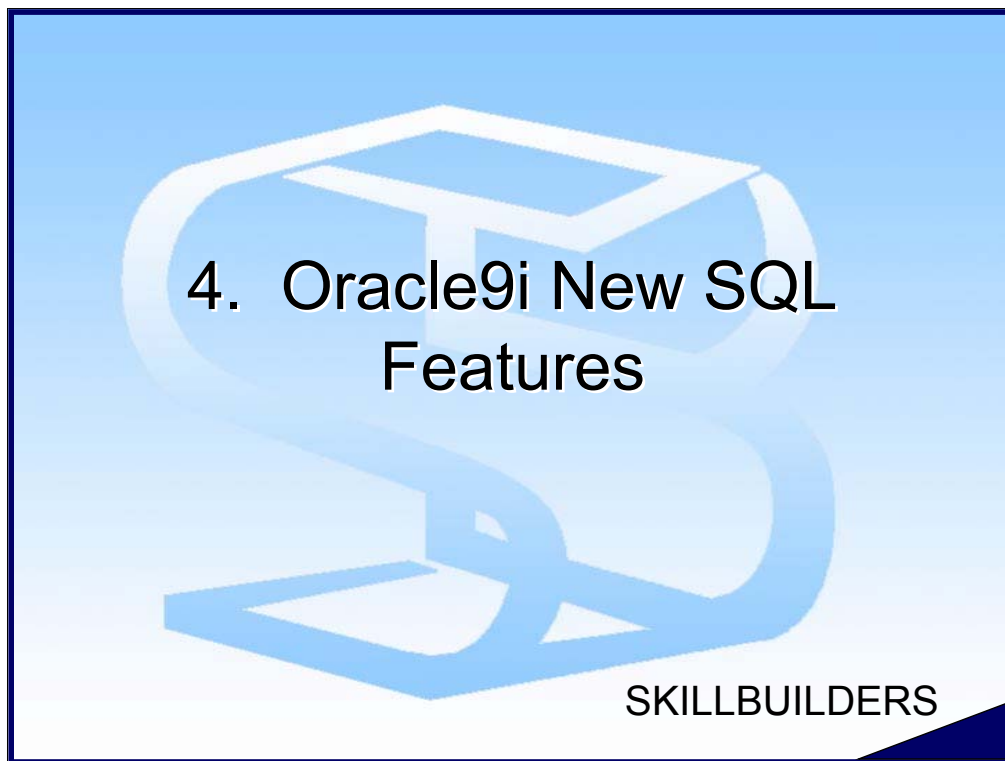


3.10

Summary

- Use External tables to query, transform and load external data
- Easy to load
 - Consider `CREATE TABLE AS SELECT`
 - `INSERT` with Sub-Select
- Convenient way to access SQL*Loader

© 2003 SkillBuilders, Inc.





4.2

9i SQL features

- Oracle9i adds:
 - `MERGE` statement
 - Multi-table `INSERT`
 - ANSI compliant joins
 - Subquery Factoring (Named subqueries)
 - `CONNECT BY` extensions
 - New functions
 - SQL CASE Expression (8.1 feature)
- Let's look at each in turn...

© 2003 SkillBuilders, Inc.




4.3

MERGE statement...

- Allows one to merge the rows of two tables
- Some columns of the merged table may also be updated simultaneously
- Since updates and inserts are performed this is known as an “upsert” function
- Processing of `MERGE` is more efficient than writing an equivalent PL/SQL routine
- Support for external tables

© 2003 SkillBuilders, Inc.

4.4

...MERGE Statement

➤ Merge external table into permanent table

```
merge into student_emails s
using (select * from student_emails_ext) e
on (s.firstname = e.firstname
     and s.lastname = e.lastname)
when matched then
    update set s.email = e.email
when not matched then
    insert (s.firstname, s.lastname, s.email)
    values(e.firstname , e.lastname, e.email);
```

External table

© 2003 SkillBuilders, Inc.

See the script `MERGE.SQL` for a working example of this code.



4.5

Multi-Table INSERT...

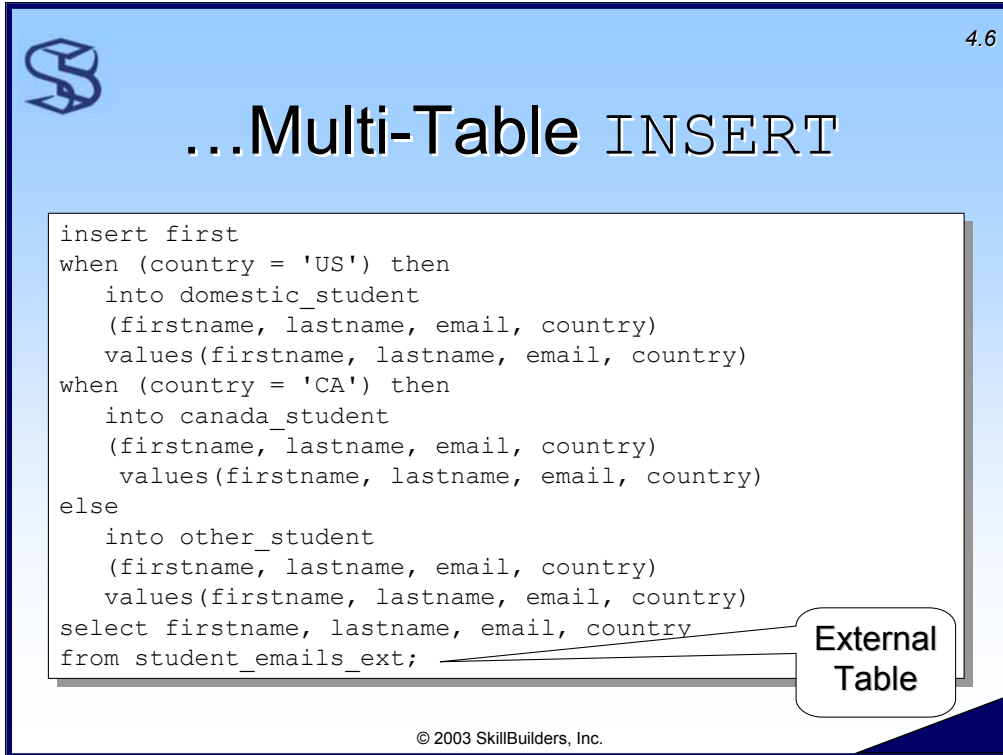
- The `INSERT` statement can now affect multiple tables
- External table support
- Syntax

```
INSERT { ALL | FIRST }  
      [ WHEN condition THEN ]  
      INTO tablename  
      [ ELSE INTO tablename ]  
select clause
```

© 2003 SkillBuilders, Inc.

The `WHEN ... THEN ... INTO` clause, if present, can repeat up to 127 times involving the same or different tables.

`ALL` will evaluate every `WHEN` clause which could result in inserts into multiple tables. `FIRST` will stop after the first `WHEN` clause evaluates to true.



4.6

...Multi-Table INSERT

```
insert first
when (country = 'US') then
  into domestic_student
  (firstname, lastname, email, country)
  values(firstname, lastname, email, country)
when (country = 'CA') then
  into canada_student
  (firstname, lastname, email, country)
  values(firstname, lastname, email, country)
else
  into other_student
  (firstname, lastname, email, country)
  values(firstname, lastname, email, country)
select firstname, lastname, email, country
from student_emails_ext;
```

External Table

© 2003 SkillBuilders, Inc.

Here is another example, using the ALL format:

```
INSERT ALL
  INTO customer (firstname, lastname)
  VALUES(firstname, lastname)
  INTO cust_history (firstname, lastname)
  VALUES(firstname, lastname)
SELECT firstname, lastname
FROM employee WHERE dept_no = 23;
```

Another example, somewhat more complex would be:

```
INSERT FIRST
  WHEN (dept_no = 111) THEN
    INTO customer (cust_no, firstname, lastname)
    VALUES(emp_no+9000, firstname, lastname)
  WHEN (dept_no = 432) THEN
    INTO cust_history (cust_no, firstname, lastname)
    VALUES(emp_no+9000, firstname, lastname)
  ELSE
    INTO customer (cust_no, firstname, lastname)
    VALUES(emp_no+9000, firstname, lastname)
SELECT emp_no, firstname, lastname, dept_no
FROM employee;
```

4.7

ANSI Compliant Joins

- Oracle9i supports ANSI/ISO SQL99 join syntax:
 - NATURAL [join-type] JOIN
 - CROSS JOIN
- Join types supported:
 - INNER
 - {LEFT | RIGHT | FULL} [OUTER]
- Provided for ANSI/ISO SQL99 conformity

© 2003 SkillBuilders, Inc.

The ANSI compliant join support in Oracle9i now allows one to write joins conformant to SQL99 syntax. You might find this syntax to be more intuitive than what was offered in earlier releases of Oracle.

The Oracle syntax notes show that there are two ways to code the ANSI join. Technique 1:

```
table_reference { CROSS JOIN | NATURAL [join_type] JOIN table_reference } }
```

Technique 2:


```
table_reference { [join_type] JOIN table_reference  
    { ON condition | USING ( column [, column]... ) } }
```

“join_type” can be :

```
{ INNER | { LEFT | RIGHT | FULL } [OUTER] }
```

The default is `INNER`. If you specify `LEFT`, `RIGHT` or `FULL`, an `OUTER` join is performed. The keyword `OUTER` is optional, but should, in my opinion, be used for clarity.

Let's look at each in turn...

4.8

Natural Join

- Natural Join uses like-named columns as the join condition


```
SELECT lastname, order_no  
FROM customer NATURAL INNER JOIN ord;
```


- Is equivalent to:

```
SELECT lastname, order_no  
FROM customer, ord  
WHERE customer.cust_no = ord.cust_no;
```

© 2003 SkillBuilders, Inc.

In a natural join the database uses the common column name(s) to perform the join. Since the `Customer` and `Ord` tables both have the `cust_no` column name in common these columns will be used to perform the join.

 The Oracle definition states “A natural join is based on *all columns* in the two tables that have the same name. It selects rows from the two tables that have equal values in the relevant columns.”

4.9

USING Clause

- Add USING clause to define join column(s):

```
select dept_name, lastname
from department INNER JOIN employee
    USING (dept_no);
```
- Add a 3rd table:

```
select dept_name, lastname, count(*)
from department INNER JOIN employee
    using (dept_no)
    natural inner join ord
group by dept_name, lastname
```

© 2003 SkillBuilders, Inc.

This join makes use of the `USING` clause to explicitly state the column on which to perform the join. Note that the `USING` clause is part of the `FROM` clause. The `USING` clause supports multiple column names: `USING (dept_no, mgr)`

You'll need to remove `NATURAL` and add the `USING` clause as shown here because it is no longer a natural join. Remember, a natural join is all like-named columns used to join the tables.

Adding a 3rd table is easy. Simply add the next joined table clause such as:

➤ `NATURAL INNER JOIN table-name.`

Additional Notes:

Use "ON" clause if there are no common columns.

```
SELECT lastname, order_no
FROM customer JOIN ord
ON customer.cust_number = ord.cust_no;
```

Outer Join syntax is supported:

```
SELECT description, nvl(quantity,0)
FROM product LEFT OUTER JOIN ord_item
USING (product_id);
```




4.10

Subquery Factoring Clause...

- Permits reusable subquery within query
 - Scope is statement
- Also called SQL “WITH” clause
- Supplies a name for a subquery block
- Can be easier to read
- Can produce better execution plan

© 2003 SkillBuilders, Inc.

The scope of the named subquery is just the statement in which it appears. It can not be used in subsequent SQL statements.

4.11

...Subquery Factoring Clause

```
WITH
average_order AS ( SELECT AVG(total_order_price)
                    AS avg_ord FROM ord ),
gold_customers AS ( SELECT cust_no FROM ord
                    WHERE total_order_price >
                        (SELECT avg_ord FROM
                         average_order) )
SELECT distinct c.cust_no, lastname, firstname,
               to_char((select * from average_order),
                       '$999.99') AS average_order
FROM customer c, gold_customers gc
WHERE c.cust_no = gc.cust_no
```

© 2003 SkillBuilders, Inc.

The example above shows the use of subquery factoring to simplify a relatively complex query. This example creates two named queries: “AVERAGE_ORDER” and “GOLD_CUSTOMERS.” Note how by naming the subqueries one can make a complex query more readable and better documented. Also note that the first name subquery block, “average_order”, is referenced in the second block, “gold_customers”.

The “old” version of this query, or at least one version of it, would look like:

```
SELECT distinct c.cust_no, lastname, firstname
FROM customer c, (SELECT cust_no FROM ord
                  WHERE total_order_price >
                     (select avg(total_order_price) from ord) ) gc
WHERE c.cust_no = gc.cust_no
ORDER BY 1;
```



4.12

CONNECT BY Extensions

- Sort same level siblings by some key

```
select lpad(' ',2*level-2) ||  
       lastname as name, hiredate  
from employee  
start with emp_no = 2  
CONNECT BY prior emp_no = mgr  
ORDER SIBLINGS BY hiredate;
```

| NAME | HIREDATE |
|------------|-----------|
| Anderson | 01-FEB-94 |
| Washington | 21-APR-94 |
| Doright | 02-AUG-94 |
| Wells | 02-AUG-94 |
| Perry | 15-MAR-95 |
| Barbee | 15-JAN-99 |
| Roger | 15-MAR-95 |
| Hall | 15-AUG-97 |

© 2003 SkillBuilders, Inc.

SIBLINGS is a new Oracle9i keyword that sorts all like-level siblings (child) rows by some key.

In this case I chose to sort by column **hiredate**.



4.13

9i Hierarchy Path...

- **SYS_CONNECT_BY_PATH** function shows path

```
select lpad(' ', 2*level-2) ||  
       lastname as name,  
       sys_connect_by_path(lastname, '/') AS path  
from employee  
start with emp_no = 2  
connect by prior  
       emp_no = mgr;
```


| NAME | PATH |
|------------|--------------------------|
| Anderson | /Anderson |
| Washington | /Anderson/Washington |
| Doright | /Anderson/Doright |
| Wells | /Anderson/Doright/Wells |
| Perry | /Anderson/Doright/Perry |
| Barbee | /Anderson/Doright/Barbee |
| Roger | /Anderson/Roger |
| Hall | /Anderson/Hall |

© 2003 SkillBuilders, Inc.

The new Oracle9i function **SYS_CONNECT_BY_PATH** reveals the hierarchy path for the specified column. It shows the entire path, from root to node.

The first argument is the column for which we want the path. The second argument is the separator.

In this example, we see the management chain starting with employee Anderson.



4.14

...9i Hierarchy Path

```

select lpad(' ',2*level-2) ||
       sys_connect_by_path(lastname, '/') as path
from employee
start with emp_no = 2
connect by prior
       emp_no = mgr;

```

| PATH |
|--------------------------|
| /Anderson |
| /Anderson/Washington |
| /Anderson/Doright |
| /Anderson/Doright/Wells |
| /Anderson/Doright/Perry |
| /Anderson/Doright/Barbee |
| /Anderson/Roger |
| /Anderson/Hall |

© 2003 SkillBuilders, Inc.

As this example shows, using the `SYS_CONNECT_BY_PATH` function in conjunction with the `LPAD` function creates a useful result.

Note that any column can be used in the `SYS_CONNECT_BY_PATH` function. For example:

```

1 select lpad(' ',2*level-2) || lastname as name,
2        sys_connect_by_path(hiredate, '/') AS path
3 from employee
4 start with emp_no = 2 connect by prior emp_no = mgr;

```

| NAME | PATH |
|------------|--------------------------------|
| Anderson | /01-FEB-94 |
| Washington | /01-FEB-94/21-APR-94 |
| Doright | /01-FEB-94/02-AUG-94 |
| Wells | /01-FEB-94/02-AUG-94/02-AUG-94 |
| Perry | /01-FEB-94/02-AUG-94/15-MAR-95 |
| Barbee | /01-FEB-94/02-AUG-94/15-JAN-99 |
| Roger | /01-FEB-94/15-MAR-95 |
| Hall | /01-FEB-94/15-AUG-97 |

However, some columns will provide more useful than others!

4.15

9i Supports Joins

➤ **CONNECT BY** and join now work!


```
select e1.lastname AS employee,  
       e2.lastname AS manager  
from employee e1, employee e2  
where e1.mgr = e2.emp_no(+)  
start with e1.emp_no = 2  
connect by prior  
           e1.emp_no = e1.mgr;
```

| EMPLOYEE | MANAGER |
|------------|----------|
| Anderson | |
| Hall | Anderson |
| Roger | Anderson |
| Doright | Anderson |
| Barbee | Doright |
| Perry | Doright |
| Wells | Doright |
| Washington | Anderson |

© 2003 SkillBuilders, Inc.

In Oracle8i and earlier, **CONNECT BY** and Join could not be used in the same query. Oracle9i now provides support for this.

In this example I incorporated a self-outer-join to the `Employee` table to pick up the lastname of the employees' manager.



4.16

New Functions...

- **ASCIIISTR** – Return argument as ASCII
 - Example:


```
SELECT ASCIIISTR('Überfluß') FROM DUAL;
```
 - Returns:


```
\00DCberflu\00DF
```
- **UNISTR** – Return argument as Unicode
 - Example:


```
SELECT UNISTR('\00DCberflu\00df') FROM DUAL;
```
 - Returns:


```
Ü b e r f l u ß
```

© 2003 SkillBuilders, Inc.

Several new functions are available with Oracle9i:


- **ASCIIISTR** takes as argument any string and returns it translated into ASCII. For characters that cannot be translated to ASCII an escaped hexadecimal format is used.
- **UNISTR** takes as argument a string in any character set and translates it into Unicode.
- **COMPOSE** takes a string as argument and returns it in Unicode with any diacritical marks combined with the preceding character to form a single character. This is because in Unicode accented characters can be represented both as a character by itself or a base character followed by the diacritical mark.
 Example:

```
SELECT COMPOSE('de' || UNISTR('\0301') || 'ja' || UNISTR('\0300')) FROM DUAL;
```


 Returns: d é j à
- **DECOMPOSE** does the opposite of **COMPOSE** by replacing single characters with diacritical marks into two characters.
 Example:

```
SELECT DECOMPOSE('déjà') FROM DUAL;
```
- **COALESCE** generalizes the **NVL** function by accepting multiple expressions as its arguments. The function will return the first non null expression or **NULL** if all expressions evaluate to null.
 Example:

```
SELECT COALESCE(dept_name, to_char(dept_no), 'HR') || ' Dept' FROM department;
```

4.17

...New Functions

- NULLIF
 - Return NULL if expressions equal
 - Else return first value

```
SELECT NULLIF('212', area_code) FROM employee;
```

© 2003 SkillBuilders, Inc.

NULLIF takes two expressions as arguments and compares them for equality. If they are equal the function returns NULL. If they are not equal the function returns the value of the first expression.



4.18

CASE Expression

➤ Easily define number and width of buckets


```
select
  sum ( case when salary between 6 and 10 then 1 else 0 end )
    as sal_6_10,
  sum ( case when salary between 11 and 15 then 1 else 0 end )
    as sal_11_15,
  sum ( case when salary between 16 and 20 then 1 else 0 end )
    as sal_16_20,
  sum ( case when salary between 21 and 25 then 1 else 0 end )
    as sal_21_25
from employee;
```

| SAL_6_10 | SAL_11_15 | SAL_16_20 | SAL_21_25 |
|----------|-----------|-----------|-----------|
| 4 | 3 | 0 | 2 |

© 2003 SkillBuilders, Inc.

Histograms can be easily created with the `CASE` expression used within an SQL `SELECT` statement (supported in 8.1.3).

In this example, to show the distribution of salary levels, I have created a histogram containing 4 buckets and each bucket has a width of 5.

4.19

Miscellaneous Features

- Enhanced LOB support
 - Functions (substr, instr, etc)

```
select substr(c2,1,1) c2_is_clob
from test
where substr(to_char(c2),1,1) = 'a'
```

- Convert LONG to LOB

```
alter table t modify (c1 clob);
```

- Scrolling cursors in OCI and JDBC
- Object types now support inheritance and type evolution

© 2003 SkillBuilders, Inc.

From the Oracle [Oracle9i Application Developer's Guide – Fundamentals Release2 \(9.2\)](#)

“The following SQL functions that accept or output character types now accept or output CLOB data as well:

||, CONCAT, INSTR, INSTRB, LENGTH, LENGTHB, LIKE, LOWER, LPAD, LTRIM, NLS_LOWER, NLS_UPPER, NVL, REPLACE, RPAD, RTRIM, SUBSTR, SUBSTRB, TRIM, UPPER

In PL/SQL, all the SQL functions listed above and the comparison operators (>, =, < and !=), and all user-defined procedures and functions, accept CLOB datatypes as parameters or output types. You can also assign a CLOB to a character variable and vice versa in PL/SQL.”

Object types now support all the features required to model OO applications namely, encapsulation, inheritance and polymorphism (dynamic method dispatch) making Oracle9i a full object-relational database system.



4.20

Summary of New SQL


- MERGE
 - Update and Insert in one statement
- Multi-Table Insert
 - Up to 127 tables
- ANSI Joins
- Subquery Factoring
- Connect By extensions
- New Functions

© 2003 SkillBuilders, Inc.

5. Oracle9i PL/SQL Features

ANSI CASE Statement
Associative Arrays
Multi-Level Collections
DBMS_METADATA
UTL_FILE
Pipelined Functions
PL/SQL Record-Based DML
Native Compilation

SKILLBUILDERS

5.2

New PL/SQL features


- Oracle9i adds:
 - ANSI compliant `CASE` statement
 - Multi-level collections
 - Pipelined Functions
 - Record-Based DML
 - Optional Native Compilation
 - `LOB` support enhancements
 - Metadata Access
 - `UTL_FILE` enhancements
- Let's look at each in turn

© 2003 SkillBuilders, Inc.

Additional Notes:

In addition to the new features listed above, Oracle9i also adds:

- PL/SQL and SQL now uses a common SQL parser so any valid SQL can now be used in PL/SQL.

5.3

ANSI Compliant CASE

- 9i R1 introduced CASE statement and CASE expression
 - Simple and Searched statements
 - Use expression within PL/SQL statement, e.g. assignment

```
[ <<label_name>> ]  
CASE case_operand  
WHEN when_operand1 THEN statements1;  
WHEN when_operand2 THEN statements2;  
.  
.  
.  
WHEN when_operandN THEN statementsN;  
[ ELSE statements; ]  
END CASE [ label_name ];
```


© 2003 SkillBuilders, Inc.

Oracle9i Release 1 introduced a PL/SQL CASE statement (Oracle8i added CASE for SQL statements). CASE can also be coded as an expression if it is desired to populate a variable with the result of the CASE logic. Examples will follow.

In all versions of the CASE statement the WHEN clause can appear any number of times. The WHEN clauses are evaluated sequentially. The 1st TRUE WHEN causes the associated statement(s) to be executed; The CASE statement then ends (execution continues after the END CASE clause). If none of the WHEN expressions is true the ELSE statement (if any) will execute.

The CASE statement raises a CASE_NOT_FOUND exception if an ELSE clause is not provided and none of the WHEN's are TRUE.

Only one THEN statement (or ELSE statement) is executed for each CASE statement. There is no "fall-through" as in the C language 'switch' statement.

5.4

Simple CASE

```
<<salary_test>>
CASE v_sal
  WHEN 12 THEN
    dbms_output.put_line('Salary is '||v_sal);
    v_sal := v_sal * 1.2 ;
    dbms_output.put_line('Salary is '||v_sal);
  WHEN 14 THEN
    dbms_output.put_line('Salary is '||v_sal);
    v_sal := v_sal * 1.15 ;
    dbms_output.put_line('Salary is '||v_sal);
  ELSE
    v_sal := v_sal * 1.1 ;
END CASE salary_test;
```

© 2003 SkillBuilders, Inc.

Here is an example of a simple CASE statement. Notes:

- The label is optional but provides good documentation.
- Each THEN can have any number of statements, each terminated with a semi-colon.
- Only the 1st TRUE THEN is executed. Control is transferred to the END CASE after the 1st TRUE THEN is executed.
- If the ELSE is not provided and none of the THEN's are TRUE, a CASE_NOT_FOUND exception is raised and control is automatically transferred to the EXCEPTION block, if coded.

Restriction: The case-operand and the when-operands can be any datatype except BLOB, BFILE, an object type, a PL/SQL record, an index-by-table, a varray, or a nested table.

See supplied script CASE1.SQL for a working example of this CASE expression.



5.5

Associative Arrays

- Formerly called Index-By tables
- An unbounded array of variables
- Use any number or character string as subscript
- Useful for lookups
- Can be passed as Procedure or Function parameter
- Only define-able at PL/SQL level

```
TYPE table_type_name IS  
  TABLE OF datatype [NOT NULL]  
  INDEX BY  
  [BINARY_INTEGER|VARCHAR2(size)];
```

© 2003 SkillBuilders, Inc.

Associative Arrays are a PL/SQL array, used to store lists of data in a PL/SQL program. This might be helpful for storing data that is repeatedly scanned – eliminating repetitive access to a database table. It can also be a useful structure for passing sets of data between PL/SQL programs. Associative Arrays were introduced with Oracle9i Release 2.


Associative arrays are arrays of variables where the variable can be a scalar type, a variable defined with **%TYPE** or a record defined with **%ROWTYPE**.

Associative arrays are unbounded, meaning that they have is no limit to the number of elements in the array. (actually, the limit is -2,147,483,647 to +2,147,483,647, or 4.3 billion rows. However, we consider them to be unbounded because you'll run out of memory before you'll ever reach the limit.)

Associate arrays are also considered sparse, in that they do not require a sequential number of rows. I.e. there can be gaps between element 1 and the second element.

Associative arrays require an index. The index can be **BINARY_INTEGER** (a number) or, with Oracle9i, a **VARCHAR2** field.

The elements in a PL/SQL array are not in any particular order and are not necessarily stored contiguously in memory. The keys used for a PL/SQL table do not have to be sequential and can be an expression as well as a constant or variable.

5.6

Record-Based DML...

➤ 9i R2 supports records in `INSERT` and `UPDATE`

```
DECLARE
    cust_rec    customer%rowtype;
BEGIN
    cust_rec.cust_no := 234;
    cust_rec.lastname := 'Anderson';
    cust_rec.firstname := 'Dave';
    INSERT INTO customer VALUES cust_rec;
EXCEPTION
    WHEN dup_val_on_index THEN
        UPDATE customer SET ROW = cust_rec
        WHERE cust_no = 234;
END;
```

© 2003 SkillBuilders, Inc.


Oracle9i R2 supports PL/SQL records in the `INSERT` and `UPDATE` DML statements.

Note that in the `INSERT`, “`cust_rec`” is NOT enclosed in parenthesis. This is required.

Note the use of the new keyword “`ROW`” in the `UPDATE` statement. This allows us to update the entire row. Unfortunately, it is not possible (yet?) to update a subset of a row using the `ROW` keyword.

You can define your own PL/SQL record (as opposed to using `%ROWTYPE`), but it must be completely compatible with the table row. I.e. You cannot define only a subset of columns.

See the supplied script `RECORD.SQL` for working code examples of record-based DML.

5.7

...Record-Based DML

```
DECLARE
  Type cust_rec_t is RECORD
    (cust_no number,
     lastname varchar2(25),
     firstname varchar2(25) );
  Type cust_tab_t is table of cust_rec_t;
  cust_tab cust_tab_t;
BEGIN
  UPDATE customer
  SET area_code = '917'
  WHERE area_code = '212'
  RETURNING cust_no, lastname, firstname
  BULK COLLECT INTO cust_tab;
  . . .
END;
```


© 2003 SkillBuilders, Inc.

Another new supported use of PL/SQL records is in the `BULK COLLECT INTO` clause.

We see in this example that we return (using bulk collect) all effected customers (customers who are updated) into the collection “cust_tab” which is based on the record “cust_rec_t”.

Note that it is still required to code all the individual column names in the `RETURNING` clause, even if you desire to return all columns. “`RETURNING *`” is not supported.

See the supplied script `RECORD.SQL` for working code examples of record-based DML.

5.8

Multi-Level Collections

- Oracle9i supports collections nested in collections
- Allows the creation of multi-dimensional Associative Arrays (formerly Index-By Tables) and `VARRAYS`

```
SQL> DECLARE
2   TYPE x IS VARRAY(30) OF INTEGER;
3   TYPE y IS VARRAY(20) OF x;
4
5   myvar_x x := x(234,42,23,42); -- init first element
6   myvar_y y := y( x(0), myvar_x, x(54,65,34,44) );
7 BEGIN
8   dbms_output.put_line(myvar_y(3)(2) );
9 END;
10 /
65
```

PL/SQL procedure successfully completed.

© 2003 SkillBuilders, Inc.

Oracle9i supports multi-level (multi-dimensional) arrays through the support of collections within collections.

In this example, “y” is now an array of 20 “x”.

To use the multi-dimensional `varray` simply define a variable of that type. You must initialize each value of the `varray` before you can access them.

The following statement declares `myvar` and initializes one `y` element containing only null `x` elements:

```
myvar y := y();
```

The following declares and initializes one `y` element containing one `x` element which only contains nulls:

```
myvar y := y( x(null, null, null) );
```

To access `myvar` as a two dimensional `varray` use the following syntax:

```
myvar(1)(1) := 123;
```

See the provided script ‘`multi_level_collections.sql`’ for a working example.




5.9

Pipelined Functions...

- Extraction, Transformation & Load (ETL) feature
- Extension to 8i Table Functions feature
 - Function is row source
- Pipelined function starts returning rows before function completes
- Caller can resume; start consuming rows
 - Can be more efficient
- Intermediate collection storage not required
 - Uses less memory

© 2003 SkillBuilders, Inc.

Oracle9i provided an enhancement to table functions called “pipelined functions.” The purpose of pipelining is efficiency. The pipelined function starts returning rows (piping rows) back to the caller before the function even completes. (Remember that in the previous example, the function passed back a complete, populated collection on the `RETURN` instruction.) In addition to passing rows back to the caller as soon as possible, the pipelined function does not require potentially large memory area to hold a populated collection – as the previous example did.

5.10

...Pipelined Functions

```
create or replace function month_generator
(p_num_months in number)
RETURN sqlMONTH_TABLEtype
PIPELINED
AS
    month_table      sqlMONTH_TABLEtype
                    := sqlMONTH_TABLEtype();
BEGIN
    for i in 1..p_num_months loop
        PIPE ROW ( add_months(sysdate, -i) );
    end loop;
    return;
END;
/
```

© 2003 SkillBuilders, Inc.


Here is an example of a pipelined function. This function performs the same function as the previous `month_generator` function, but is more efficient because:

- the `PIPE ROW` instruction “pipes” rows back to the caller immediately – before the function fully completes, and
- the function requires less memory because it does not have to fully populate the collection and pass the collection back to the caller, it simply passes the rows back as it creates them (with the `PIPE ROW` statement).

Notables:

- Use the `PIPELINED` keyword in the function header to define a pipelined function.
- Use the `PIPE ROW` statement to send the rows back to the caller.
- Do NOT code any return value on the `RETURN` instruction.

See supplied script `TABLEFUNC2.SQL` for a working example.

5.11

Native Compilation

- Optionally compile PL/SQL into C
 - Linked into Oracle kernel
- Computationally intensive routines will benefit
 - Tests show up to 2 times as fast
- DBA sets up
 - Update \$ORACLE_HOME/plsql/spnc_makefile.mk
 - Set init.ora parameters

```
ALTER SESSION SET plsql_compiler_flags = native;  
CREATE PROCEDURE test AS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Hello');  
END;
```

© 2003 SkillBuilders, Inc.

Oracle9i provides optional native compilation for PL/SQL programs. When native compilation is used, Oracle will convert the PL/SQL into C, compile the C into object code, then linked into the Oracle executable (kernel). When the routine is invoked, Oracle simply calls the linked subroutine.

Programs that are reliant on lots of computations will run faster when compiled natively. Tests show up to 2x as fast as interpreted routines. (See asktom.oracle.com and search on “oracle9i pl/sql native compile” for some good examples.)

The DBA will need to setup the server for native compilation before it can be used.

Additional Notes

Refer to the init.ora parameters `plsql_native_make_utility` and `plsql_native_make_file_name` when preparing your server for native compilation.

Query the data dictionary view `USER_STORED_SETTINGS` to determine the compilation method. The `PARAM_VALUE` column will contain `NATIVE` if native compilation was used to create the object.



5.12

LOB Support

➤ Comparison operations on LOBs

```
1 declare
2     v1 clob := 'abc';
3 begin
4     If v1 = 'abc' then null; end if;
5* end;
SQL> /
```

PL/SQL procedure successfully completed.

➤ With ALTER to change LONG to LOB, now have transparent path to LOBs

© 2003 SkillBuilders, Inc.



5.13


UTL_FILE Improvements

- Searches Oracle DIRECTORY
 - Replaces init parameter UTL_FILE_DIR
 - Change w/o restarting database

```
1* create directory seq_files as 'C:\Oracle\oradata\external'  
SQL> /  
Directory created.
```

- Read and write binary files
 - PUT_RAW / GET_RAW
- Remove files
 - FREMOVE
- More...

© 2003 SkillBuilders, Inc.

5.14

Metadata Access...

- DBMS_METADATA package allows access to object information

```
SELECT
  DBMS_METADATA.GET_DDL('TABLE',
                        'CUSTOMER', 'SCOTT') FROM DUAL;
SELECT
  DBMS_METADATA.GET_XML('TABLE',
                        'CUSTOMER', 'SCOTT') FROM DUAL;
```

- See next page for output...

© 2003 SkillBuilders, Inc.

The first query allows one to retrieve the DDL that can create the `Customer` table. The second query retrieves the `Customer` table metadata in XML format.

See the supplied script `METADATA.SQL` for a working example.



5.15

...Metadata Access

```
SQL> set linesize 4000
SQL> Set long 4000
SQL> set heading off
SQL> SELECT
  2   DBMS_METADATA.GET_DDL('TABLE','CUSTOMER','DAVE')
  3   FROM DUAL;
```

```
CREATE TABLE "DAVE"."CUSTOMER"
(
  "CUST_NO" NUMBER(*,0),
  "LASTNAME" VARCHAR2(20) NOT NULL ENABLE,
  "FIRSTNAME" VARCHAR2(15) NOT NULL ENABLE,
  "MIDINIT" VARCHAR2(1),
  "CITY" VARCHAR2(20),
  "STATE" VARCHAR2(2),
  . . .
```

© 2003 SkillBuilders, Inc.

The complete example:

```
LOCAL> set heading off
```

```
LOCAL> 1
```

```
1* SELECT DBMS_METADATA.GET_DDL('TABLE', 'CUSTOMER', 'DAVE') FROM DUAL
LOCAL> /
```

```
CREATE TABLE "DAVE"."CUSTOMER"
(
  "CUST_NO" NUMBER(*,0),
  "LASTNAME" VARCHAR2(20) NOT NULL ENABLE,
  "FIRSTNAME" VARCHAR2(15) NOT NULL ENABLE,
  "MIDINIT" VARCHAR2(1),
  "STREET" VARCHAR2(30),
  "CITY" VARCHAR2(20),
  "STATE" VARCHAR2(2),
  "ZIP" VARCHAR2(5),
  "ZIP_4" VARCHAR2(4),
  "AREA_CODE" VARCHAR2(3),
  "PHONE" VARCHAR2(8),
  "COMPANY_NAME" VARCHAR2(50),
  PRIMARY KEY ("CUST_NO")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "TOOLS" ENABLE )
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "TOOLS"
```



5.16

Summary of PL/SQL Features

- ANSI compliant `CASE` statement
- Associate Arrays
- Multi-level collections
- Record-Based DML
- Pipelined Functions
- Native Compilation
- Enhanced `LOB` support
- `UTL_FILE` Enhancements
- Metadata Access
- And, . . . Common Parser
 - PL/SQL now uses same SQL parser

© 2003 SkillBuilders, Inc.

6. New Datatypes

Datetime Datatypes
INTERVAL Datatypes
XMLType Datatype

SKILLBUILDERS




6.2

Intro to 9i Datetime...

- 9i DATETIME datatype provides support for fractional seconds and time zones
- Datetime datatype stores some or all of:
 - Year, Month, Day
 - Hour, Minute Second, Fractional second
 - Time Zone hour displacement
 - Displacement from UTC
 - Time Zone minute displacement
 - Time Zone region name
 - Time Zone abbreviation

© 2003 SkillBuilders, Inc.

Oracle9i provides new datatypes for dealing with date and time data. There are three new “datetime” datatypes, so-called because they contain both date and time data. The primary enhancement is the ability to save and manage time zone data. This could be an assist for those of us writing applications used in multiple time zones.

6.3

...Intro to 9i Datetime

- 3 new datetime datatypes
- `TIMESTAMP`
 - `DATE` with up to 9 digits of fractional seconds
- `TIMESTAMP WITH TIME ZONE`
 - `Timestamp` with timezone preserved
- `TIMESTAMP WITH LOCAL TIME ZONE`
 - `TIMESTAMP` normalized to DB time zone
 - Converts to DB time zone on `INSERT`
 - Converts to Session time zone on `SELECT`

© 2003 SkillBuilders, Inc.

Three new datetime datatypes are available. Examples and more details are provided on the following pages.

- **Timestamp** gives us fractional seconds, but no time zone support. 6 digits of fractional seconds are provided by default. 9 digits is the maximum. Override by specifying number of digits, e.g. `TIMESTAMP(2)`.
- **Timestamp with Time Zone (TSTZ)** gives us fractional seconds and time zone support. `TSTZ` always remembers the time zone inserted with the datetime data, and makes no adjustments to the datetime data.
- **Timestamp with Local Time Zone (TSLTZ)** gives us fractional seconds and time zone support. `TSLTZ` never remembers the time zone inserted, but adjusts the datetime from the session time zone to the database time zone during the insert operation. Oracle will again adjust the datetime data when selecting, changing from database time zone to the time zone of the session the query is running in.



6.4


Datetime Example...

- TSTZ saves time zone
 - Useful when need to remember TZ of input
- TSLTZ: INSERTS data in the DB time zone
 - Converts time to *DB* on way in
 - Converts time to *session* on way out

```
CREATE TABLE new_types (  
  a  TIMESTAMP,  
  b  TIMESTAMP WITH TIME ZONE  
  c  TIMESTAMP WITH LOCAL TIME ZONE);
```

© 2003 SkillBuilders, Inc.

See the supplied script `DATETIME.SQL` for a working example of a PL/SQL program that utilizes datetime data.

6.5

...Datetime Example

➤ Session1 forced into Dublin time zone:

```
alter session set time_zone = 'Europe/Dublin';

insert into new_types
values ('26-APR-02 04.34.29.123456 AM',
       '26-APR-02 04.34.29.123456 AM Africa/Cairo',
       '26-APR-02 04.34.29.123456 AM');
```


➤ Session 2 in US Eastern time zone:

```
select * from new_types
26-APR-02 04.34.29.123456 AM
26-APR-02 04.34.29.123456 AM AFRICA/CAIRO
25-APR-02 10.34.29.123456 PM
```

© 2003 SkillBuilders, Inc.

Notice the following:

- Time zone information is irrelevant to **TIMESTAMP data** (column 1)
- Time zone of the inserting session is preserved for **TIMESTAMP WITH TIME ZONE data**.
- Time zone of the inserting session is lost for **TIMESTAMP WITH LOCAL TIME ZONE data**. However, the time is adjusted to the session time zone as it is selected from the database.

6.6

Datetime Related Items

- New PL/SQL literals
 - `TIMESTAMP`
 - `DATE`
- New functions
 - `dbtimezone` and `sessiontimezone`
 - `TO_TIMESTAMP`
 - `TO_TIMESTAMP_TZ`
- Math
 - Addition and subtraction of datetime data
 - Datetime – Datetime produces `INTERVAL` type

© 2003 SkillBuilders, Inc.

9i provides two new ANSI-compliant literals for use with `TIMESTAMP` and `DATE` data. We will see examples of these literals on the following pages.

Similar to the `TO_DATE` function, Oracle provides the `TO_TIMESTAMP` and `TO_TIMESTAMP_TZ` functions for converting character strings to `TIMESTAMP` data. Refer the the Oracle SQL Reference for more information.

Like `DATE` data, we will need – and Oracle provides – the ability to add and subtract from datetime data. Refer the the Oracle SQL Reference (9.2), section “Basic Elements of Oracle SQL, 2 of 10” for more information.

Additional Notes:

- Oracle9i also introduces 2 new `INTERVAL` datatypes to support the concept of time intervals. A time interval is any period of time, e.g. 1 year, 8 hours, or 7 days + 4 hours.
- `INTERVAL YEAR TO MONTH` Datatype - stores a period of time using the `YEAR` and `MONTH` datetime fields.
- `INTERVAL DAY TO SECOND` Datatype - stores a period of time in terms of days, hours, minutes, and seconds.
- Refer to the SQL Reference for more information on the `INTERVAL` datatypes.



6.7

INTERVAL Types


➤ Elapsed time between two timestamps

- INTERVAL YEAR [precision] TO MONTH
- INTERVAL DAY [precision] TO SECOND [precision]

```
SQL> declare
2   t1   timestamp := current_timestamp + 1 ;
3   i1   INTERVAL DAY TO SECOND;
4   begin
5       i1 := t1 - current_timestamp;
6       dbms_output.put_line(i1);
7   end;
8   /
+00 23:59:59.133000
```

PL/SQL procedure successfully completed.

© 2003 SkillBuilders, Inc.

6.8


XML in the Database

- 9i introduces `XMLType`
- Store native XML in a table column
- `XMLType` and related functions collectively called “XML DB”
- `XMLType` supports
 - XPath searches
 - XSL Transformations
 - OLAP Functions
 - More...

© 2003 SkillBuilders, Inc.

Oracle provides a data type and related functions that they collectively call “XML DB”.

Oracle9i R1 introduced the ability to store native XML documents in the database. This is accomplished with the new data type called `XMLType`. We can perform XPath searches, XSL transformations and more on `XMLType` data.

6.9

Using XMLType


- XMLType is an Object Type
- Use in PL/SQL and CREATE TABLE

```
create table all_my_ddl
(ddl_id number primary key,
ddl      xmltype );
```

```
DECLARE
    v_ddl XMLType;
    v_text varchar2(100);
BEGIN
    . . .
```

© 2003 SkillBuilders, Inc.

XMLType is a valid Oracle data type and can be used in the CREATE TABLE command and in PL/SQL blocks.

6.10

Inserting XML

➤ Use the XMLTYPE constructor (function) to insert XML

```
insert into all_my_ddl
values (1,
xmltype(
  (SELECT DBMS_METADATA.GET_XML('TABLE',
                                'CUSTOMER',
                                'STUDENT1')
   FROM DUAL)
)
);
```


© 2003 SkillBuilders, Inc.

Oracle 9.2 provides a constructor method called “XMLTYPE” to INSERT XML into a XMLType column.

In this example I use the output of the GET_XML procedure call to generate the XML document to be inserted into my table.

Note that Oracle 9.1 provided a static method called XMLType.createxml() for the same purpose:

```
insert into all_my_ddl
values (2,
xmltype.createxml( (SELECT DBMS_METADATA.GET_XML('TABLE', 'ORD',
'STUDENT1') FROM DUAL) ) );
```

6.11

Extracting XML...

➤ Extract value of a single node:


```
select
  extractvalue(ddl, '/ROWSET/ROW/TABLE_T/TS_NAME')
  AS tsname
from all_my_ddl

TSNAME
-----
USERS
```

© 2003 SkillBuilders, Inc.

The `EXTRACTVALUE` function can be used to display the text values of a single (non-repeating) node. Pass the column name and slash-delimited path in XPath notation.

In this example we can see that the DDL saved in my table indicates the table was created in the `USERS` tablespace.

6.12

...Extracting XML

➤ Use `EXTRACT` to display value in repeating node


```
select extract(ddl,  
  '/ROWSET/ROW/TABLE_T/COL_LIST/COL_LIST_ITEM/NAME')  
  AS col_name  
from all_my_ddl  
where extractvalue(ddl,  
  '/ROWSET/ROW/TABLE_T/SCHEMA_OBJ/NAME')  
  = 'CUSTOMER';
```

| COL_NAME |
|------------------------|
| ----- |
| <NAME>CUST_NO</NAME> |
| <NAME>LASTNAME</NAME> |
| <NAME>FIRSTNAME</NAME> |

© 2003 SkillBuilders, Inc.

When a collection of nodes can be returned it is necessary to use `EXTRACT` rather than `EXTRACTVALUE`.

In this example the `NAME` node repeats N times – once for each column in a table.


6.13

PL/SQL and XML

```
DECLARE
    v_ddl  XMLType;
    v_text varchar2(100);
BEGIN
    select ddl into v_ddl
    from all_my_ddl
    where ddl_id = 1;
    v_text := v_ddl.extract
        ('/ROWSET/ROW/TABLE_T/TS_NAME/text()'
        ).getstringval;
    dbms_output.put_line(v_text);
END;
```

© 2003 SkillBuilders, Inc.

PL/SQL has been enhanced to support access to XMLType data.

6.14

Other XML Features

- Create index on `XMLType` node
- Many more functions
 - `UPDATEXML` for `UPDATE` statements
 - `XMLTransform` for XSLT transformations
- For more information see
 - Oracle XML Database Developers Guide, 9.2

© 2003 SkillBuilders, Inc.

Oracle9i R2 includes robust support for storing native XML in the database. For example, indexing a node in the XML document is a possibility. There are also many more functions available than the ones shown in this section.


See the following Oracle manuals for more information:

XML API Reference - XDK and Oracle XML DB

XML Database Developer's Guide - Oracle XML DB

XML Developer's Kits Guide - XDK

See the supplied script `XML1.SQL` for working XML demonstration code.

6.15

Datatype Summary

- Datetime datatypes include
 - `TIMESTAMP`
 - `TIMESTAMP WITH TIME ZONE`
 - “Remembers” time zone inputted
 - `TIMESTAMP WITH LOCAL TIME ZONE`
 - Normalizes time
- **INTERVAL datatypes**
 - Holds time duration (elapsed time)
 - Use in arithmetic operations involving timestamps, intervals
- **XMLType**
 - Native XML support in table column

© 2003 SkillBuilders, Inc.

Database applications that support cross-time zone data will often benefit from the new `TIMESTAMP` datatypes.

Refer to the Oracle Globalization Guide for some good examples of using datetime data.

7. Resumable Space Management

Concepts
Enabling
Monitoring

SKILLBUILDERS



7.2

Introduction to Resumable Space Management...

- Manages space allocation failures
 - Reduces rework
- Suspends a transaction
 - Intercepts error
 - Allows DBA to correct
- Allows a transaction to resume
 - Automatically restarts
- Transaction can be suspended and resumed multiple times during execution

© 2003 SkillBuilders, Inc.

Most DBAs have run into space allocation errors before; unable to extend a segment within a tablespace, maximum number of extents reached, and exceeding a tablespace quota.

In earlier releases of Oracle, all of these conditions were easy to correct, but the main problem was that the transaction causing one of these conditions to occur had to be RESTARTED from the beginning, resulting in lost time. This can become very frustrating for everyone involved, especially if the problem happens again after the transaction has been restarted for a second or third time.

With RSM, the transaction that has encountered the space allocation error is suspended for a period of time - allowing you to fix the suspend condition and eventually allowing the transaction to resume. This is a very valuable enhancement that could result in a huge amount of saved time especially in systems where there are large data loads or large transactions coupled with vague data requirements.



7.3

...Introduction to Resumable Space Management


- RSM protection provided at the session level
- For a RSM enabled session:
 - Automatic transaction suspension after encountering a space allocation error
 - Transaction will appear to hang
 - No message to user
 - Data dictionary views are available to the DBA for diagnosing the problem
 - Automatic transaction continuance when error is corrected

© 2003 SkillBuilders, Inc.

If RSM is configured for a particular session and a suspend condition is encountered, the user's session will appear to hang. It is not initially obvious whether the problem is a locking problem, a performance problem, or a statement suspension.

Once such a problem is reported, you should go through your normal diagnostic process to determine the cause of the reported problem. Besides checking for locking and performance issues, in Oracle9i you can also check for the possibility that a statement has been suspended. This is done with several data dictionary views which are covered later in the Monitoring RSM section.

After correcting the problem, the user's transaction or statement resumes processing until it either completes or is suspended again.

7.4

Errors Handled

- Errors handled include
 - Unable to extend segment
 - ORA-1650, ORA-1653, ORA-1654
 - Max extents reached
 - ORA-1628, ORA-1631, ORA-1632
 - Exceeding a tablespace quota
 - ORA-1536

© 2003 SkillBuilders, Inc.

RSM handles a number of space allocation errors. The following errors can be suspended to allow you time to correct a space allocation problem so that the transaction may resume without losing any work.

Unable to extend segment


```
ORA-01650 unable to extend rollback segment xxx by xxx in tablespace xxx
ORA-01653 unable to extend table xxx by xxx in tablespace xxx
ORA-01654 unable to extend index xxx by xxx in tablespace xxx
```

Max extents reached

```
ORA-01628 max # extents xxx reached for rollback segment string xxx
ORA-01631 max # extents xxx reached in table xxx
ORA-01632 max # extents xxx reached in index xxx
```

Exceeding a tablespace quota

```
ORA-01536 space quota exceeded for tablespace xxx
```

7.5

Enabling RSM

- Done at the session level
- Consider ON LOGON trigger
- Use ALTER SESSION or dbms_resumable
- Example: Enable RSM, suspend transaction for 3 hours:

```
alter session enable resumable timeout 10800  
name 'Update of hr table';
```

```
SQL> alter session disable resumable;  
Session altered.
```

- Must have RESUMABLE system privilege

© 2003 SkillBuilders, Inc.


Enabling RSM is accomplished at the session level with either the `ALTER SESSION` command or programmatically with the `DBMS_RESUMABLE` supplied package.

The `TIMEOUT` option specifies in seconds the amount of time a transaction will suspend once it has encountered a space allocation error. If `TIMEOUT` is not defined it defaults to two hours. If the `TIMEOUT` period has expired without the space allocation error being corrected the transaction is terminated with the error message that caused the suspension along with the following error message:

```
ORA-30032: the suspended (resumable) statement has timed out
```

Before an Oracle user can take advantage of the Resumable Space Management feature, they must have been granted the `RESUMABLE` system privilege.

```
SQL> grant resumable to app_developer;  
Grant succeeded.
```



7.6

Monitoring RSM...

- Considered a DBA responsibility
- New data Dictionary Views
 - DBA_RESUMABLE, USER_RESUMABLE

```
SQL> select ul.username, r1.name, status, timeout
       2  from dba_resumable r1, dba_users ul
       3  where ul.user_id = r1.user_id;
```


| USERNAME | NAME | STATUS | TIMEOUT |
|----------|--------------------|-----------|---------|
| ----- | ----- | ----- | ----- |
| SYSTEM | Update of hr table | SUSPENDED | 10800 |

© 2003 SkillBuilders, Inc.

There are several places you can monitor the suspension of a transaction. You can query the new `DBA_RESUMABLE` and `USER_RESUMABLE` show sessions in resumable mode and their status. `V$SESSION_WAIT`, `V$SYSTEM_EVENT`, and `V$SESSION_EVENT` also show information about sessions that have been suspended along with other non-related database events.

Some important columns in the `DBA_RESUMABLE` view:

- `USERNAME` – User id of suspended session.
- `SESSION_ID` – Session id of the statement is resumable mode.
- `SQL_TEXT` – The first 1000 bytes of the SQL statement.
- `NAME` – The comment given when resumable mode was enabled.
- `STATUS` – The status of the SQL statement. `SUSPENDED` or `NORMAL`.
- `TIMEOUT` – The timeout duration in seconds if the SQL statement should suspend.
- `START_TIME` – The time the SQL statement started.
- `ERROR_MSG` – The error message of the error encountered.
- `SUSPEND_TIME` – The time when the SQL statement was suspended.
- `RESTART_TIME` – The time when the SQL statement was resumed.

7.7

...Monitoring RSM

- An entry is recorded in the alert log for when a session:
 - Suspends
 - Identifies the error that caused the suspension
 - Resumes
 - Times out

Mon May 20 15:29:23 2002

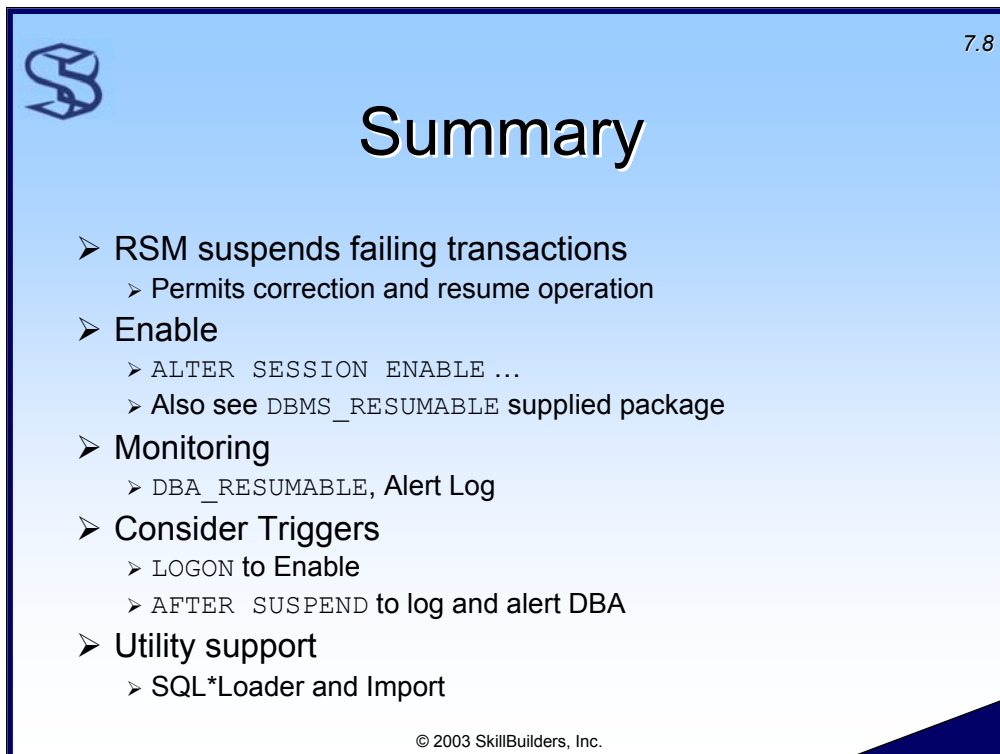
statement in resumable session 'User ORDER_USER(40), Session 7, Instance 1' was suspended due to

ORA-01653: unable to extend table ORDER_USER.ORDER_SUMMARY by 16 in tablespace ORDER_TBS

© 2003 SkillBuilders, Inc.

Sample Alert.log Entries:

```
Mon May 13 14:39:26 2002
statement in resumable session 'User APP_DEVELOPER(29), Session 7,
Instance 1' was suspended due to
ORA-01536: space quota exceeded for tablespace 'CUSTOMER_TBS'
...
Mon May 13 14:46:42 2002
statement in resumable session 'User APP_DEVELOPER(29), Session 7,
Instance 1' was resumed
...
Tue May 14 15:08:53 2002
statement in resumable session 'User ORDER_USER(38), Session 7, Instance
1' was suspended due to
ORA-01653: unable to extend table ORDER_USER.ORDER_SUMMARY by 16 in
tablespace ORDER_TBS
...
Tue May 14 17:08:53 2002
statement in resumable session 'User ORDER_USER(38), Session 7, Instance
1' was timed out
```



7.8

Summary

- RSM suspends failing transactions
 - Permits correction and resume operation
- Enable
 - `ALTER SESSION ENABLE ...`
 - Also see `DBMS_RESUMABLE` supplied package
- Monitoring
 - `DBA_RESUMABLE`, Alert Log
- Consider Triggers
 - `LOGON` to Enable
 - `AFTER SUSPEND` to log and alert DBA
- Utility support
 - SQL*Loader and Import

© 2003 SkillBuilders, Inc.

Resumable Space Management (RSM) can save a lot of time when dealing with long running transactions that fail due to a storage allocation error. With RSM, a transaction can be suspended giving you time to correct the problem, at which time the transaction resumes.

Before using RSM, a session must be enabled for it by executing either the `ALTER SESSION` command or the `DBMS_RESUMABLE` package.

DBA's can use `DBA_RESUMABLE` to monitor for suspended transactions.


Triggers can be used to help automate RSM. For example, a `LOGON` trigger could be used to enable RSM for specific sessions. The new `AFTER SUSPEND` trigger can be used to log and possibly alert the DBA to the existence of a suspended transaction.

Both SQL*Loader and import have been enhanced to support RSM.

8. Tuning Enhancements for Developers

V\$SQL_PLAN Stats
Dynamic Sampling
Cursor Sharing Enhancement
Forced Rewrite
Parallel DML
New Hints

SKILLBUILDERS


8.2

Runtime Plan & Statistics

- Query `V$SQL_PLAN` to see run time access path
- Query `V$SQL_PLAN_STATISTICS` to see runtime statistics (R2)

© 2003 SkillBuilders, Inc.

The runtime access path of SQL are now saved in case one would want to query them. This allows one to look at the plan without having to rerun the SQL for this purpose. The `V$SQL_PLAN` table holds the access plans.



8.3

V\$SQL_PLAN

- Use V\$SQL_PLAN view to see cached plan
 - Stores the actual execution plan information
 - Has similar columns to the PLAN_TABLE

```
SQL> SELECT operation, options, object_name
2      FROM v$sql_plan
3      WHERE address = '79DCCF68'
4      AND hash_value = 2966840198;
```

| OPERATION | OPTIONS | OBJECT_NAME |
|------------------|---------|-------------|
| ----- | ----- | ----- |
| SELECT STATEMENT | | |
| HASH JOIN | | |
| TABLE ACCESS | FULL | DEPARTMENT |
| TABLE ACCESS | FULL | EMPLOYEE |

© 2003 SkillBuilders, Inc.


Cached execution plans are similar to the plans generated by the `EXPLAIN PLAN` utility. However, cached execution plans are the actual execution plans for statements that have already run and are stored in the shared pool. Once a statement ages out of the shared pool, its cached execution plan also ages out.

Let's assume I ran this query:

```
select /*davea*/ *
      from employee e, department d
      where e.dept_no = d.dept_no
```

To get the `ADDRESS` and `HASH_VALUE`, execute:

```
SELECT sql_text, address, hash_value
FROM v$sql
WHERE sql_text like '%davea%'
```

8.4

Dynamic Sampling

- Compile time sampling of data to determine selectivity and volume
- Useful when statistics old or non-existent
- Useful for queries executed many times or with long execution time
- Recursive SQL issued to read random sample of data blocks
- Set `optimizer_dynamic_sampling` parameter or `dynamic_sampling` hint
 - 0 – no sampling
 - 1 through 10 – Various levels of sampling; 10 most aggressive

© 2003 SkillBuilders, Inc.

Dynamic sampling is the Oracle9i R2 feature that allows the cost-based optimizer to sample data blocks at compile time to generate selectivity and cardinality (i.e. volume or number of rows) statistics. This can lead to better execution plans if the collected statistics are out of date or non-existent.


Since there is a performance hit at compile time, queries that execute many times or have a long execution time (in relation to the compile duration) may benefit from dynamic sampling.

How does it work? Oracle will issue a recursive (Oracle-generated) SQL statement to read a random sampling of data blocks – at compile time – to collect the statistics.

Dynamic sampling is controlled by the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter and the `DYNAMIC_SAMPLING` hint. Set to 0 to turn off the feature. Set to 1 through 10 to control the level of sampling.

Supplemental Notes

1 is the default `OPTIMIZER_DYNAMIC_SAMPLING`. In essence, this causes Oracle to sample if there is an unanalyzed table in the query.

8.5

Cursor Sharing...

- CBO automatically rewrite queries to incorporate bind variables
- Convert

```
select *    from customer
where area_code = '212';
```


- To this:

```
select *    from customer
where area_code = :SYS_B_0;
```

© 2003 SkillBuilders, Inc.

The level of matching between two SQL statements can be controlled with the `CURSOR_SHARING` parameter. This parameter was available in Oracle8i but only with the valid values of `EXACT` and `FORCE`. `SIMILAR` is a new valid value for `CURSOR_SHARING` in Oracle9i and later.

If the `CURSOR_SHARING` parameter has been set to `SIMILAR` or `FORCE` then the SQL statements shown in the slide would be considered the same and would not have to be reparsed. However, while this can provide better performance in many cases, it is not always a good thing. For example, what if there are a significantly higher number of customers in area code 401? The same plan may not be optimal for both statements.

8.6

Cursor Sharing...

- Valid values:
 - **EXACT** (default)
 - See previous page
 - **SIMILAR** (New with 9i)
 - Identical in all aspects other than literal values
 - Execution plan found is considered for use
 - **FORCE**
 - Similar to **SIMILAR**
 - Execution plan found is always used
- Also see new `CURSOR_SHARING_EXACT` hint

© 2003 SkillBuilders, Inc.

EXACT forces Oracle to only use existing parsed information in the shared pool if it exactly matches the statement it is currently being compared to. The rules for what makes two statements identical are discussed on the previous page.

SIMILAR is new to Oracle9i. It allows Oracle to use information in the shared pool if two statements only differ in the literal values they use. It still must make a determination whether or not to use the execution plan found in the shared pool.

FORCE is similar to **SIMILAR** except that it will always use the execution plan found in the shared pool for the matching statement.



8.7

Forced Rewrite

- Force the rewrite of queries to use materialized views

```
SQL> alter session set query_rewrite_enabled = force;  
Session altered.
```

© 2003 SkillBuilders, Inc.



8.8

Skip Scanning...


- Prior to 9i, index usage required leading column
- Skip Scanning allows index w/o leading column
- Requires the use of CBO
- See `PLAN_TABLE.OPERATION` and `OPTIONS`
 - `OPERATION = INDEX, OPTIONS = SKIP SCAN`

© 2003 SkillBuilders, Inc.

Prior to Oracle9i, a predicate in the `WHERE` clause had to reference the high order column of the index for the optimizer to choose that index. The new Skip Scanning feature allows an index to be used even if the predicate does NOT reference the high order (leading) column. The Cost-Based Optimizer must be used to take advantage of `SKIP SCAN`.

Query the `OPERATION` and `OPTIONS` columns of the `PLAN_TABLE` to see if `SKIP SCAN` is used:

- `OPERATION = INDEX, OPTIONS = SKIP SCAN`

8.9

...Skip Scanning

- Assume the existence of this index:

```
create index fullname on customer(lastname, firstname)
```

- Query can use SKIP SCAN on index:

```
select *  
  from customer  
 where firstname = 'Dave';
```

© 2003 SkillBuilders, Inc.

This example illustrates the benefit of the Oracle9i `SKIP SCAN` index technique. A query that does not include the leading, high order column of an index can now use that index to reduce query processing time.



8.10

Miscellaneous...

- R2 supports parallel DML on non-partitioned tables
- `FIRST_ROWS_n` hint
 - Optimize for specified number of rows
 - Also see `first_rows_n` Initialization parameter
- CBO peeks at bind variable values on hard parse
 - Provides ability to determine selectivity

© 2003 SkillBuilders, Inc.



8.11

...Miscellaneous

- I/O, memory and CPU costs are now taken into account when calculating cost by optimizer
- Outline Editing
- MONITORING on indexes to determine use
- Other new hints
 - R1: NL_AJ, NL_SJ, FACT, NO_FACT
 - R2: EXPAND_GSET_TO_UNION

© 2003 SkillBuilders, Inc.



8.12

Going Away

- 9.2.x.x.x will be the last release to support rule-based optimization
- Oracle Trace is deprecated
 - Use STATSPACK

© 2003 SkillBuilders, Inc.



8.13

Summary


- `v$sql_plan` and `v$sql_plan_statistics` provide window to actual plan and stats
- Dynamic sampling overcomes old or non-existent stats
- Cursor sharing can help reduce parsing
- Forced rewrite can help take advantage of materialized views
- Parallel DML and more...

© 2003 SkillBuilders, Inc.

9. Security for Developers

Fine Grained Access Control
n-Tier Proxy Authentication
Label Security
Data Encryption

SKILLBUILDERS

9.2

8i FGAC Review...

- 8i gave us Fine Grained Access Control, AKA
 - Virtual Private Database
 - Row Level Security
 - `DBMS_RLS`
- Security logic embedded in database
- Ability to tack on a `WHERE` clause at run-time
 - Sort of a “dynamic view”

© 2003 SkillBuilders, Inc.

Oracle8i provided a security feature called Fine Grained Access Control (FGAC). This feature is also sometimes called Virtual Private Database, row level security and `DBMS_RLS` (the supplied package that implements the feature).

FGAC allows us to put simple or complex security logic in the database, as opposed to the client or application server. The basic functionality of FGAC is to, based on the circumstances that your logic determines, add a `WHERE` clause to the query (`SELECT`, `UPDATE`, `INSERT` and `DELETE` are all supported) being executed, thereby restricting the user to a set of rows.

FGAC can reduce the number of views needed in your database if you tend to create different views for different users. You may also be creating different stored procedures and/or triggers for different user groups. So you may be able to reduce the amount of procedural objects you have to maintain as well.

Some environments reduce the complexity of managing security by allowing shared user accounts, e.g. a group of employees all log in with the same username. FGAC reduces (maybe eliminates) the need for this by significantly reducing the complexity of administering security.



9.3


...8i FGAC Review

- Application contexts provide additional intelligence to the security policy
 - Variables set at logon give session context
 - Security logic checks context
 - Applies `WHERE` clause based on context
- FGAC can reduce:
 - Number of views needed
 - Code maintenance
 - The need for shared user accounts

© 2003 SkillBuilders, Inc.

A key component of FGAC is “application context”. An application context is simply a database object (create with `CREATE CONTEXT`) that has a PL/SQL package bound to it. The procedures in the package can assign values to any number of variables associated with the context. This is typically done at login time via a `LOGON` trigger. Later in the life of the session, when a query is made against a table that has a security policy tied to it, the security policy (PL/SQL routine) checks the context variable values and makes decisions based on those values.

For example, the application context may set a variable that informs the security policy that the user is a manager, and is entitled to see all data, or the user is not a manager, and is only entitled to see data related to his or her employment.

9.4

9i FGAC Enhancements

- FGAC / VPD enhancements
 - Oracle Policy Manager
 - GUI tool to manage FGAC
 - Global Application Context
 - Application context persists across sessions
 - Useful for Web app support, where each page is new session
 - Partitioned Fine Grained Access Control
 - FGAC by application, instead of just user


© 2003 SkillBuilders, Inc.

Oracle9i Release 1 supplied several enhancements to FGAC. These include:

Oracle Policy Manager. This is a graphical tool that can be used to simplify FGAC management. It is part of the Oracle Enterprise Manager (OEM) tool.

Global Application Context. This extends the functionality of application context to persist across sessions. This is particularly helpful in Web applications where a session may terminate and restart by visiting different Web pages.

Partitioned Fine Grained Access Control. This extends FGAC to allow different applications to have different security policies – even on the same object. Tom Kyte (asktom.oracle.com) describes it this way: “Partitioned fine grained access control is the ability to have FGAC by application.”

9.5

n-Tier Proxy Authentication

- “Application Server logs in as itself, but on behalf of another user” (1)

```
ALTER USER application_Server_userid GRANT  
CONNECT THROUGH proxy_user WITH roles;
```

- Support for auditing, FGAC
- Enhancements
 - Java JDBC support
 - Identification through X.509 certificates or Distinguished Names (DN)
 - More...

© 2003 SkillBuilders, Inc.

n-Tier proxy authentication allows a middle-tier server logon on behalf of another user, using its own logon credentials. It is accomplished in large part with this new form of the `ALTER USER` command:


```
ALTER USER application_Server_userid GRANT CONNECT THROUGH proxy_user WITH  
roles;
```

Proxy authentication allows the server to apply FGAC security policies and auditing to the proxy account, and accurately report who caused the activity. This feature was introduced with Oracle8i. With Oracle9i, Oracle has added support for Java (formerly, only C or C++ using OCI could make use of the feature). Support for X.509 certificates and Distinguished Names has also been added.

Resources:

(1) Tom Kyte, asktom.oracle.com

Oracle Advanced Security Administrators Guide

9.6

Other 9i Security Features

- Label Security
 - VPD without programming
 - Control with Oracle Policy Manager
- Fine grained auditing
 - Conditional auditing of `SELECT` statements
- Data encryption enhancements
 - FIPS-140 certified random number generator
 - See the `DBMS_OBFUSCATION_TOOLKIT` package

© 2003 SkillBuilders, Inc.

Oracle9i introduces Label Security, an out-of-the-box FGAC solution. i.e. It takes the programming out of fine grained access control and virtual private database. The Oracle Policy Manager, part of Oracle Enterprise Manager, is a graphical tool for controlling and managing Label Security.

Fine Grained auditing (FGA) is a new feature in Oracle9i that allows you to conditionally audit `SELECT` statements. We'll discuss this feature in greater detail later in this module.

Release 1 also raised the bar on the data encryption capabilities available with the `DBMS_OBFUSCATION_TOOLKIT` supplied package by supplying a better, FIPS-140 certified, random number generator for encryption keys. (FIPS is a Federal Information Processing Standard, a US government security standard.)



9.7

Security Summary

- Fine Grained Access Control
 - GUI tool
 - Partitioned Application Contexts
- n-Tier Proxy Authentication
 - Java Support
- Label Security
 - VPD in a box
- Data Encryption
 - FIPS certified Random Key generator

© 2003 SkillBuilders, Inc.

10. Index Enhancements for Developers

IOTs and bitmap indexes
Bitmap Join Indexes

SKILLBUILDERS



10.2

Bitmap Join Indexes

- Reduces amount of data to be joined during query


```
create bitmap index emp_dept_bidx on emp( d.dname )
from emp e, dept d
where e.deptno = d.deptno;

select count(*)
from emp, dept
where emp.deptno = dept.deptno
and dept.dname = 'SALES'
```

- Conceptually adding column from DEPT to EMP

© 2003 SkillBuilders, Inc.

The bitmap join index pre-computes the join and stores the result in a bitmap. In our example, the bitmap index will contain all the `dept.dname` values and a bitmap that maps to the rowids in the `dept` table.

10.3


IOTs and Bitmap Indexes...

- IOTs now support bitmap indexes
- Requires the use of a mapping table
 - Use `MAPPING TABLE` clause on `CREATE TABLE`
- Mapping Table facts:
 - Associates a bit from the bitmap index to the `ROWID` from the IOT
 - Stored in same tablespace as parent IOT
 - Mapping table name controlled by Oracle
 - E.g. `SYS_IOT_MAP_30526`

© 2003 SkillBuilders, Inc.

Indexes allowed on IOTs may now be bitmap indexes as well as B-tree indexes. This requires the use of a “mapping table.” The mapping table associates a bit from the bitmap index to the `ROWID` from the IOT.

The name of the mapping table is generated by Oracle. The mapping table is placed in the same tablespace as the IOT.



10.4

...IOTs and Bitmap Indexes

```

create table phones
( area_code      number
,exchange        number
,last_four       number
,constraint phone_pk
primary key(area_code, exchange, last_four) )
organization index
mapping table tablespace main_ts;

create bitmap index phone_bitmap on
phones(last_four);

```

© 2003 SkillBuilders, Inc.

A mapping table can also be added to an existing IOT:

```
SQL> ALTER TABLE countries MOVE MAPPING TABLE TABLESPACE ts_data1;
Table altered.
```

Yes, you code “MOVE” not “ADD”, when adding a mapping table to an existing IOT!

You can see the effect of the MAPPING TABLE clause with this query:

```
LOCAL> select table_name , iot_type
2   from user_tables
3  where iot_type is not null
4  /
```

| TABLE_NAME | IOT_TYPE |
|-------------------|-------------|
| PHONES | IOT |
| SYS_IOT_MAP_30526 | IOT_MAPPING |



10.5

Summary

- Bitmap join indexes can reduce / eliminate join processing
 - Especially good for Data Warehouse environments
- IOTs now support bitmap indexes

© 2003 SkillBuilders, Inc.



10.6

Presentation Summary

➤ Questions???

➤ Thanks for Listening!

© 2003 SkillBuilders, Inc.