CERN School of Computing

Advanced Database Features

Zornitsa Zaharieva

CERN

Accelerators and Beams Department
Controls Group, Data Management Section
AB-CO-DM

09-SEP-2005

CERN School of Computing

Contents

Oracle Data Dictionary Views

- Store all the information that is used to manage the objects in the database
- Source of valuable information for developers and db users
- USER_* , ALL_* , DBA_*

SYS.DICTIONARY	lists all objects that make up the data dictionary
SYS.USER_TS_QUOTAS	lists all of the tablespaces and how much can be used/is used
SYS.USER_OBJECTS	lists objects created in the user's schema
SYS.USER_TABLES	lists tables created in the user's schema
SYS.USER_VIEWS	lists views created in the user's schema
SYS.USER_CONSTRAINTS	lists all the constraints (e.g. Check, PK, FK, Unique) created on user objects
SYS.USER_SYS_PRIVS	lists system privileges
SYS.USER_ROLE_PRIVS	lists roles granted to the user

Access Remote Data – Database Link

- A database link is an object in the *local* database that allows you to access objects on a *remote* database
- Database link syntax:

```
CREATE DATABASE LINK remote_connect
CONNECT TO user_account IDENTIFIED BY password
USING 'connect_string';
```

Name of the link

Service name - gives connection details for the communication protocol, host name, database name; stored in a file (tnsnames.ora)
example – devdb, edmsdb, cerndb1

Name of the account in the remote database

Password for the account

- Access tables/views over a database link (view USER_DB_LINKS)

```
SELECT * FROM emp@remote_connect;
```

- Restrictions to the queries that are executed using db link
  : avoid *CONNECT BY, START WITH, PRIOR*

CERN School of Computing

Synonyms

- Synonyms are aliases for tables, views, sequences
  (view USER_SYNONYMS)

- Create synonym syntax for a remote table/view

```
CREATE  SYNONYM emp_syn
    FOR  emp@remote_connect;
```

- Use synonyms in order to
  : simplify queries
  : achieve location transparency - hide the exact physical location of a database object from the user (application)
  : simplify application maintenance

- Example of accessing a table over a db link with a synonym

```
SELECT  * FROM emp_syn;
```

CERN School of Computing

Grant / Revoke Privileges

- DBAs can grant/revoke any administrative privilege
  create session, create tables, views, sequences, etc.
- The developer can grant/revoke privileges on the objects they own
  select / insert / update / delete / execute
- Access can be granted on db objects, tables or columns

```
GRANT SELECT ON orders_seq TO alice;
GRANT UPDATE (salary) ON employees TO alice;
```

- Check USER_TAB_PRIVS and USER_COL_PRIVS
- Accessing an object in another schema

```
SELECT  *  FROM  hr.employees;
```

- Use synonyms to achieve location transparency and simplify application maintenance

CERN School of Computing

Views

- A stored SQL statement that defines a virtual table
- If the data in tables emp or dept change, the view displays the changes immediately
- Create view syntax example (using an inline view in the query)

```
CREATE OR REPLACE VIEW  v_emp_salary AS
    SELECT   e.ename     ,e.job        ,e.sal
             ,s.maxsal   ,s.deptno   ,d.name
      FROM    emp e       ,dept d
             ,(SELECT  MAX(sal) maxsal, deptno
                 FROM  emp
                GROUP BY deptno) s
     WHERE   e.deptno = s.deptno
       AND   e.deptno = d.deptno
    ORDER  BY e.deptno, e.sal DESC;
```

CERN School of Computing

Updatable Views

- If a view is based on a single underlying table you can insert, update, delete rows in the view
- Some restrictions
  - : cannot insert, if the underlying table has any *NOT NULL* columns that do not appear in the view
  - : cannot insert, update, delete if the view contains *GROUP BY*, *DISTINCT*
- You can insert into a multitable view, if the underlying tables are *key-preserved*

Key-preserved table: a view contains enough columns from a table to identify the primary key for that table

CERN School of Computing

Views – Benefits and Typical Usage

- To make complex queries easy
  : hide joins, sub-queries, order behind the view
  : provide different representations of the same data

- To restrict data access
  : restrict the columns which can be queried
  : restrict the rows that queries may return
  : restrict the rows and columns that may be modified

- To provide abstract interface for data independence
  : users (applications) form their queries on the basis of the views, no need to know the physical tables
  : if the tables change, no need to rewrite the queries

CERN School of Computing

Indexes – Main Purpose

- To enforce uniqueness
  : when a PRIMARY KEY or UNIQUE constraint is created, Oracle automatically creates an index to enforce the uniqueness of the indexed columns

- To improve performance
  : when a query can use an index, the performance of the query may dramatically improve

- By default Oracle creates B-tree index (view USER_INDEXES)

```
CREATE INDEX ord_customers_i ON orders (customer_id);
```

Index-Organized Tables (IOT)

- IOT stores all of the table's data in the index
- A normal index only stores the indexed columns in the index
- IOT syntax

```
CREATE table orders_iot (    order_id       NUMBER
                            ,order_date  DATE
                                ...........
                            ,CONSTRAINT order_iot_pk PRIMARY KEY (order_id)
                          )
 ORGANIZATION INDEX;
```

- Use if always accessing the table's data by its primary key
- Efficient if the primary key constitutes a large part of the table's columns

Bitmap Index

- Appropriate when
  : low cardinality columns are used as limiting conditions in a query (columns with few discrete values)
  : if the data is infrequently updated, since they add to the cost of all data manipulation transactions against the tables they index

CREATE BITMAP INDEX customer_region_i ON customers (region);

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
	arried	central	female	bracket_3

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

Function-based Indexes

- An index created after applying a function to a column

```
CREATE INDEX customer_name_i ON sales ( UPPER(customer_name) );
```

- Bitmap indexes can also be function-based

```
CREATE BITMAP INDEX customer_region_i ON customers ( UPPER(region) );
```

Note: The more indexes there are on a table, the longer all inserts, updates and deletes will take.

CERN School of Computing

Materialized Views

- Copies (replicas) of data, based upon queries.
- Materialized views can be
  - : local copies of remote tables that use distributed data
  - : summary tables for aggregating data
- Refreshes can be done automatically
- Known as ‘snapshot’ in previous versions of Oracle rdbms.
- In comparison to other database objects that can be used for data aggregation
  - : table created from a table – fast response time, but does not follow changes of data in the parent tables
  - : view – follow changes of data in the parent tables, but slow time response to complex queries with ‘big’ parent tables

Materialized Views - Syntax

Section 1 : header with the name of the mview

Section 2 : setting storage parameters

Section 3 : setting the refresh options

Section 4 : the query that the mview will use

```
(1) CREATE MATERIALIZED VIEW my_mview
(2) TABLESPACE DATA01
(3) REFRESH FORCE
    START WITH SysDate NEXT SysDate+1/24
    WITH PRIMARY KEY
(4) ENABLE QUERY REWRITE
AS
subquery;
```

Note: The mviews can be used to alter query execution paths – query rewrite

Note: Indexes can be created on the mview, for example a primary key

```
CREATE UNIQUE INDEX my_mview_pk ON my_mview (column1 ASC)
TABLESPACE  INDX01;
```

CERN School of Computing

Materialized Views – Refresh Process

- Refresh
  : on commit
  : on demand – changes will occur only after a manual refresh
  : automatic refresh

```
START WITH SysDate NEXT SysDate+1/24
```

- Manual refresh

```
execute DBMS_MVIEWS.REFRESH('my_mview', 'c');
  c – complete
  f  - fast
  ? – force
```

- Refresh options
  : fast - only if there is a match between a row in the mview directly to a row in the base table(s); uses mview logs
  : complete – completely re-creates the mviews
  : force – uses fast refresh if available, otherwise a complete one

CERN School of Computing

Refresh Groups

- Used to enforce referential integrity among materialized views
- Create a refresh group

```
DBMS_REFRESH.MAKE ( name      => 'my_group'
                   ,list      => 'my_mview1', 'my_mview2'
                   ,next_date => SysDate
                   ,interval  => 'SysDate+1/24');
```

- Add a mview to a group - DBMS_REFRESH.ADD
- Remove a mview from a group - DBMS_REFRESH.SUBTRACT
- Alter refresh schedule - DBMS_REFRESH.CHANGE

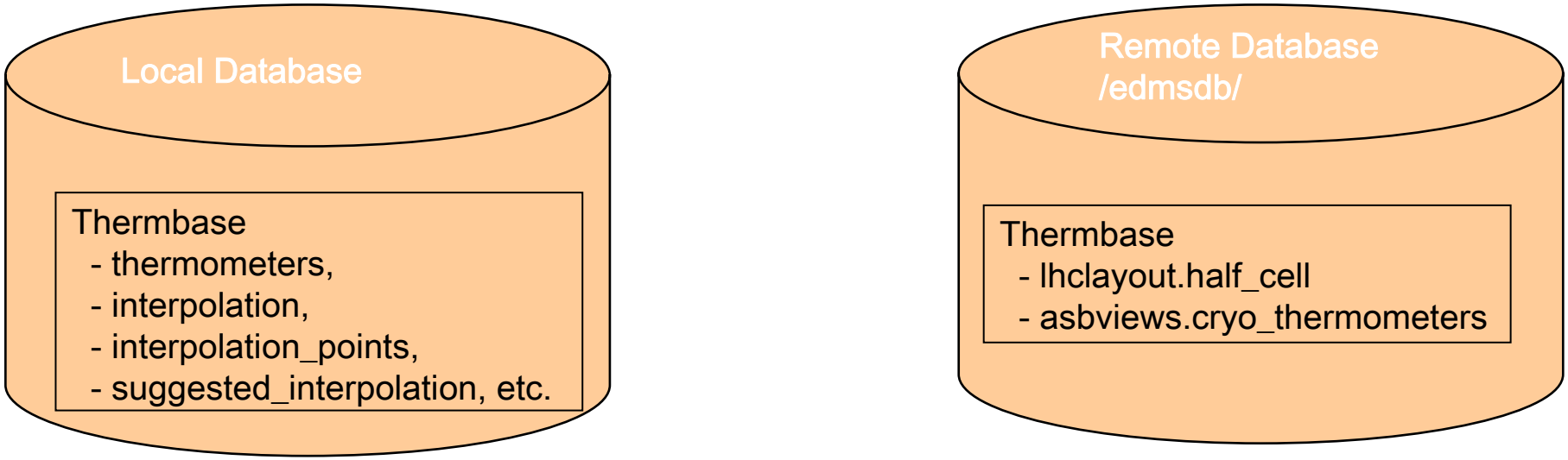
Note: While the refresh_group is performing the refresh on the mviews, the data in the mviews is still available!

Real World Example

In order to configure some of the Front End Computers in the controls systems for the LHC, they have to be 'fed' with cryogenic thermometers settings . The data that they need is split between several database schemas on different databases.

How can I solve the problem?

Step 1: I need to access data on a remote database
Step 2: I need to use materialized views to hold the aggregated data that I need

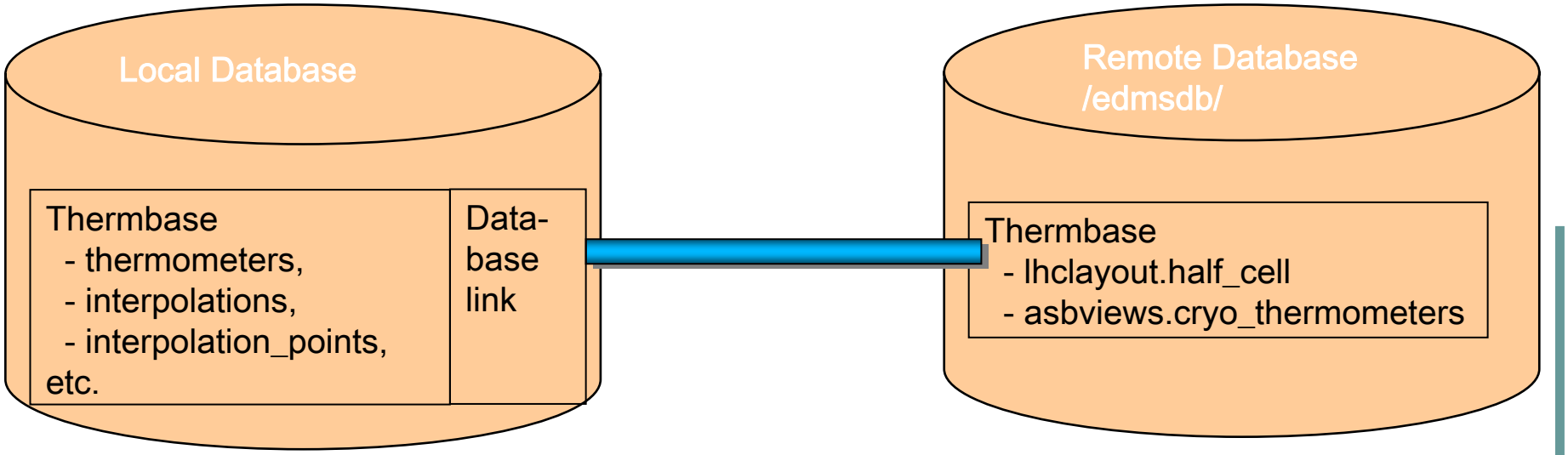


Real World Example

Step 1: Access data on a remote database - Use a database link and synonyms

```
CREATE DATABASE LINK edmsdb_link
 CONNECT TO thermbase IDENTIFIED BY password
 USING 'edmsdb';

 CREATE SYNONYM cryo_thermometers
       FOR asbviews.cryo_thermometers@edmsdb_link;
```



CERN School of Computing

Real World Example

Step 2: Use of a materialized view to hold the aggregated data that I need.

```
CREATE MATERIALIZED VIEW mtf_thermometers
refresh force
with rowid
as
SELECT  part_id   ,description
        ,tag       ,top_assembly
        ,slot_id   ,SUBSTR(top_assembly, 3, 5) as system
        ,SUBSTR(slot_id, INSTR(slot_id,'.')+1) as location
 FROM  cryo_thermometers
ORDER BY part_id;

CREATE UNIQUE UNDEX mtf_thermometers_pk ON mtf_thermometers (part_id ASC)
TABLESPACE thermbase_idx;

EXECUTE DBMS_REFRESH.MAKE (  name       => 'mtf_thermometers_group'
                            ,list       => 'mtf_thermometers'
                            ,next_date => SysDate
                            ,interval   => 'SysDate+1/24');
```

CERN School of Computing

Materialized Views Benefits

- Make complex queries easy
- Provide abstract interface for data independence
- Significant time performance improvement compared to views
- If the master table is not available, the materialized view will still have the data
- The data will be automatically updated every hour, once it is scheduled
- Using a refresh group – no ‘down time’ – the user can access the data even during the time the refresh is executed

CERN School of Computing

Partitioning

- Partitioning is the key concept to ensure the scalability of a database to a very large size

  : data warehouses (large DBs storing with data accumulated over many years, optimized for read-only data analysis)

  : online systems - periodic data acquisition

- Tables and indexes can be split into smaller and easily manageable pieces called partitions

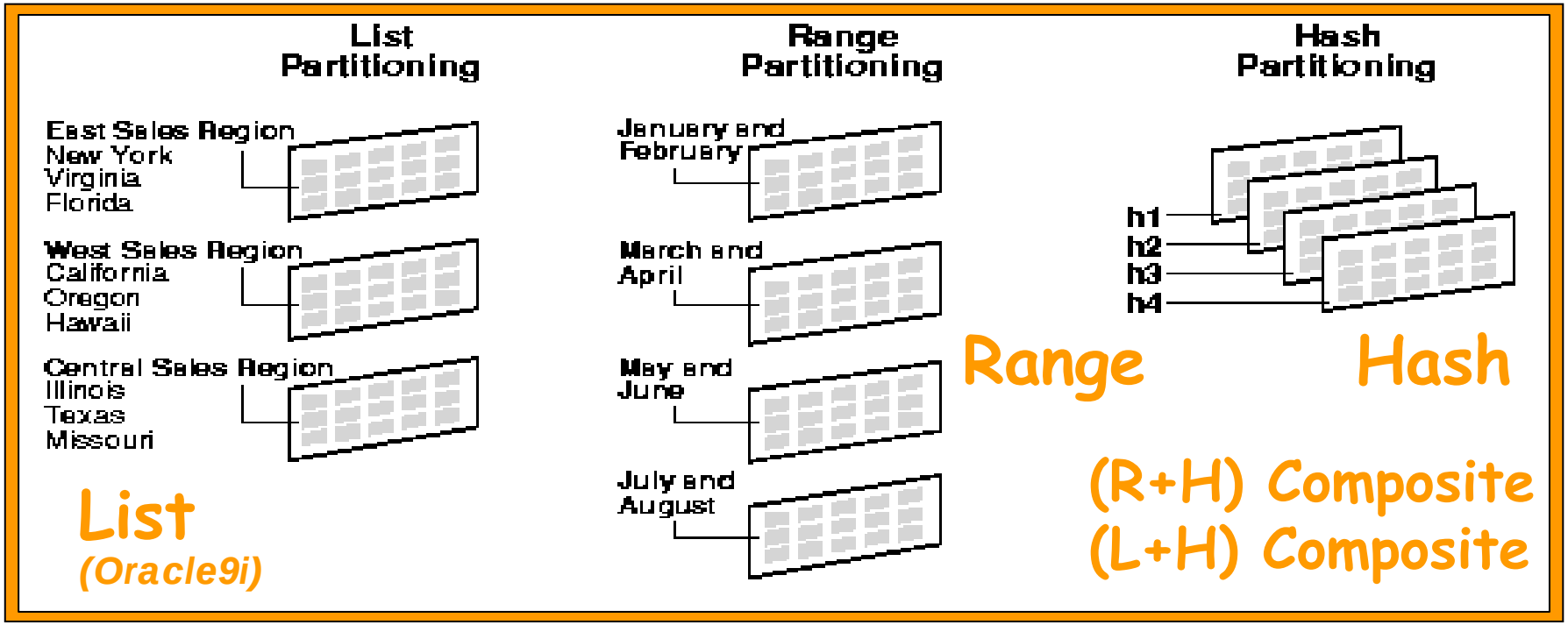
- Query performance improvement

  : queries are restricted to the relevant partitions of the table

CERN School of Computing

Partitioning - Types

- Range - partition by predefined ranges of continuous values
- Hash - partition according to hashing algorithm applied by Oracle
- Composite - e.g. range-partition by key1, hash sub-partition by key2
- List - partition by lists of predefined discrete values



Partitioning Benefits – Partition Pruning

- *Loading data into a table partitioned by date range*

```
INSERT INTO sales ( …, sale_date, … )
  VALUES ( …, TO_DATE('3-MARCH-2001','dd-mon-yyyy'), … );
```

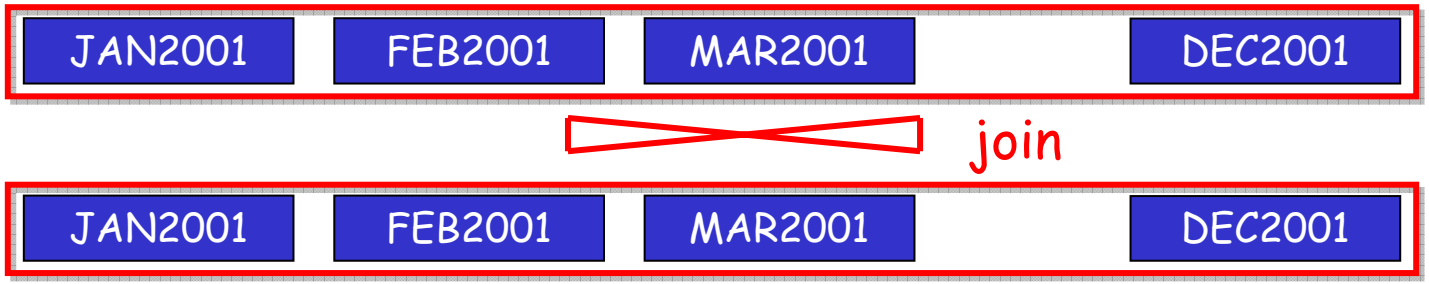
- *Querying data from a table partitioned by date range*


```
SELECT … FROM sales
  WHERE sales_date = TO_DATE ('14-DEC-2001','dd-mon-yyyy');
```

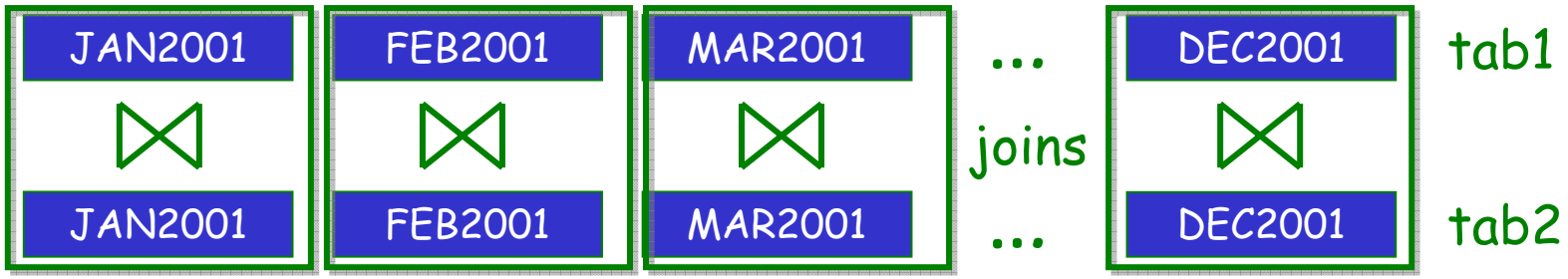
Partitioning Benefits – Partition Wise Joins

```
SELECT … FROM tab1, tab2 WHERE tab1.key = tab2.key AND …
```

- Without partitioning: global join (query time ~ N x N)



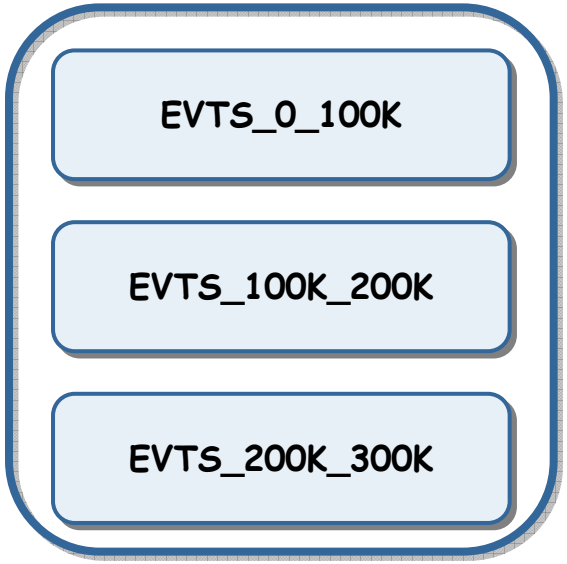
- With partitioning: local joins (query time ~ N)



Partitioning Example – Range Partitioning

```
CREATE TABLE  events
   (event_id   NUMBER(10),
    event_data BLOB)
PARTITION BY RANGE(event_id) (
  PARTITION evts_0_100k
    VALUES LESS THAN (100000)
    TABLESPACE tsa,
  PARTITION evts_100k_200k
    VALUES LESS THAN (200000)
    TABLESPACE tsb,
  PARTITION evts_200k_300k
    VALUES LESS THAN (300000)
    TABLESPACE tsc
);
```

Assigning different partitions to different table spaces further simplifies data management operations (export/backup) and allows parallel I/O on different file systems.



CERN School of Computing

PL/SQL Introduction

- Procedural Language superset of the Structured Query Language
- Used to
  : codify the business rules through creation of stored procedures and packages
  : execute pieces of code when triggered by a database event
  : add programming logic to the execution of SQL commands
- Provides high-level language features
  : complex data types
  : data encapsulation
  : modular programming

CERN School of Computing

PL/SQL Introduction

- Proprietary to Oracle RDBMS
- Integrated with the Oracle database server
  - : code can be stored in the database
  - : integral part of the database schema
  - : shared and accessible by other users
  - : execution of the code is very fast, since everything is done inside the database

CERN School of Computing

PL/SQL Blocks

- Structured PL/SQL code
- Anonymous and stored blocks
- Structure of a PL/SQL block

: **Declarations** – defines and initializes the variables and cursors used in the block

: **Executable commands** – uses flow control commands (conditional statements, loops) to execute different commands and assign values to the declared variables

: **Exception Handling** – provides customized handling of error conditions

```
DECLARE
  <declaration section>
BEGIN
  <executable commands>
EXCEPTION
  <exception handling>
END;
```

CERN School of Computing

PL/SQL Datatypes

- PL/SQL datatypes include
  - : all of the valid SQL datatypes

```
l_dept_number        NUMBER(3);
```

  - : complex datatypes (e.g. record, table, varray)
- Anchored type declarations allow to refer to the type of another object
  - : %TYPE: references type of a variable or a database column
  - : %ROWTYPE: references type of a record structure, table row or a cursor

```
l_dept_number        dept.deptnb%TYPE
```

- Advantages of anchored declaration
  - : the actual type does not need to be known
  - : in case the referenced type had changed the program using anchored declaration will be recompiled automatically

CERN School of Computing

PL/SQL Records

- Record type is a composite type
  : similar to C structure
- Declaration of a record

```
dept_rec      dept%ROWTYPE;

TYPE type_dept_emp_rec   IS RECORD (
                                      dept_no      dept.deptno%TYPE
                                     ,dept_name dept.dname%TYPE
                                     ,emp_name emp.ename%TYPE
                                     ,emp_job      emp.job%TYPE
                                    );
dept_emp_rec  IS type_dept_emp_rec;
```

- Using record variable to read a row from a table

```
SELECT  deptno, dname, loc
    INTO   dept_rec
  FROM  dept
WHERE  deptno = 30;
```

PL/SQL Conditional Control, Loops

- Conditional Control

  : IF, ELSE, ELSIF statements
  : CASE

- Loops

  : Simple loop

```
LOOP
   EXIT WHEN condition;
   <statements>
END LOOP;
```

  : WHILE loop

```
WHILE condition LOOP
   <statements>
END LOOP;
```

  : FOR loop - numeric range

```
FOR I IN 1..10 LOOP
   <statements>
END LOOP;
```

CERN School of Computing

PL/SQL Cursors

- Every SQL query produces a result set
  : a set of rows that answers the query
  : set can have 0 or more rows
- PL/SQL program can read the result set using a cursor
- A simple cursor example

```
CURSOR simple_dept_cursor IS
  SELECT deptno, dname, loc
    FROM dept;
```

- More complex example of a cursor – passing a parameter

```
CURSOR complex_dept_cursor (p_depnumber IN NUMBER) IS
  SELECT deptno, dname, loc
    FROM dept
  WHERE deptno > p_depnumber;
```

CERN School of Computing

Using Cursors

- Basic use

  : OPEN
  : FETCH
  : CLOSE

- Cursor's attributes - determine the status of a cursor

  : %NOTFOUND
  : %FOUND
  : %ISOPEN
  : %ROWCOUNT

```
DECLARE
   l_dept_number      dept.deptnp%TYPE;
   CURSOR dept_cursor (p_dept_number IN NUMBER) IS
          SELECT deptno, loc
            FROM dept
          WHERE deptno > p_dept_number;

   dept_record  dept_cursor%ROWTYPE;

BEGIN
   l_dept_number   := 20;

   OPEN dept_cursor (l_dept_number);
   LOOP
      FETCH dept_cursor INTO dept_record;
      EXIT WHEN dept_cursor%NOTFOUND;

      do_something (dept_record.deptno, dept_record.loc);
   END LOOP;
   CLOSE dept_cursor;

EXCEPTION
  WHEN OTHERS THEN
     RAISE_APPLICATION_ERROR(-20001, 'Error with departments');
END;
```

Using Cursors

- Cursor FOR loop

```
DECLARE
   l_dept_number      dept.deptnp%TYPE;
   CURSOR dept_cursor (p_dept_number IN NUMBER) IS
         SELECT deptno, loc
           FROM dept
         WHERE deptno > p_dep_number;
BEGIN
   l_dept_number   := 20;
   FOR dummy_record IN dept_cursor(l_dep_number) LOOP
      do_something (dummy_record.deptno, dummy_record.loc);
    END LOOP;
EXCEPTION
  WHEN OTHERS THEN
     RAISE_APPLICATION_ERROR(-20001, 'Error with
departments');
END;
```

CERN School of Computing

PL/SQL Procedures and Functions

- Procedures and functions are named blocks
  - : anonymous block with a header
  - : can be stored in the database
- The name of the block allows to invoke it from other blocks or recursively
- Procedures and functions can be invoked with arguments
- Functions return a value
- Values may also be returned in the arguments of a procedure

CERN School of Computing

PL/SQL Procedures and Functions

- The header specifies
  - : name and parameter list
  - : return type (function headers)
  - : any of the parameters can have a default value
  - : modes - IN, OUT, IN OUT

- Function example

```
CREATE FUNCTION get_department_no (
                                        p_dept_name IN VARCHAR2 := null
                                      ) RETURN NUMBER
IS
DECLARE
  - - - - - - - -
BEGIN
 - - - - - - - - -
  RETURN(l_dept_no);
EXCEPTION
  - - - - - - - - -
END;
```

- Procedure example

```
CREATE PROCEDURE department_change (
                                        p_dept_number IN        NUMBER
                                        p_new_name    IN OUT    VARCHAR2
                                      )

AS
DECLARE
  ..............
END;
```

CERN School of Computing

PL/SQL Packages

- Packages group logically related PL/SQL procedures, functions, variables
  : similar idea to OO Class
- A package consist of two parts
  : specification - public interface
  : body - private implementation
  : both have structure based on the generic PL/SQL block
- Package state persist for the duration of the database session

PL/SQL Packages – Advantages of Using Them

- Packages promote modern development style
  - : modularity
  - : encapsulation of data and functionality
  - : clear specifications independent of the implementation
- Possibility to use global variables
- Better performance
  - : packages are loaded once for a given session

Oracle Supplied PL/SQL Packages

- Many PL/SQL packages are provided within the Oracle Server
- Extend the functionality of the database
- Some example of such packages:

: DBMS_JOB - for scheduling tasks

: DBMS_OUTPUT - display messages to the session output device

: UTL_HTTP - makes HTTP(S) callouts
Note: can be used for accessing a web-service from the database

: PL/SQL web toolkit (HTP, HTF, OWA_UTIL, etc.)
Note: can be used for building web-based interfaces
e.g. https://edms.cern.ch

CERN School of Computing

Triggers

- Triggers are stored procedures that execute automatically when something (event) happens in the database:
  - : data modification (INSERT, UPDATE or DELETE)
  - : schema modification
  - : system event (user logon/logoff)
- Types of triggers
  - : row-level triggers
  - : statement-level triggers
  - : BEFORE and AFTER triggers
  - : INSTEAD OF triggers (used for views)
  - : schema triggers
  - : database-level triggers

PL/SQL Triggers

- Trigger action can be any type of Oracle stored procedure
- PL/SQL trigger body is built like a PL/SQL procedure
- The type of the triggering event can be determined inside the trigger using conditional predicators

  IF inserting THEN … END IF;

- Old and new row values are accessible via :old and :new qualifiers
- If for each row clause is used the trigger will be a row-level one

PL/SQL Trigger Example

```
TRIGGER THERMOMETERS_BEF_INS_ROW
BEFORE INSERT ON thermometers
FOR EACH ROW
DECLARE
  thermometers_declared          NUMBER;
  thermometers_allowed           NUMBER;
  thermometers_in_batch          NUMBER;
  thermometer_number_error     EXCEPTION;
BEGIN

   SELECT COUNT(*)
     INTO thermometers_declared
     FROM thermometers
    WHERE batch_batch_key = :new.batch_batch_key;

   SELECT num_of_block - NVL(reject_number,0)
     INTO thermometers_in_batch
     FROM batches
    WHERE batch_key = :new.batch_batch_key;

   thermometers_allowed := thermometers_in_batch - thermometers_declared;

   IF (thermometers_allowed <= 0) THEN
     RAISE thermometer_number_error;
   END IF;
EXCEPTION
  WHEN thermometer_number_error THEN
    RAISE_APPLICATION_ERROR(-20001, 'The number of thermometers declared cannot exceed the number of thermometers in that batch');
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20002, 'Error from THERMOMETERS_BEF_INS_ROW');

END;
```

Development Tools

- Oracle provided tools
  : SQL* Plus
  : JDeveloper

- Benthic Software - http://www.benthicsoftware.com/
  : Golden
  : PL/Edit
  : GoldView
  : at CERN - G:\Applications\Benthic\Benthic_license_CERN.html

- CAST - http://www.castsoftware.com/
  : SQL Code-Builder

CERN School of Computing

References

[1] Feuerstein, S., Pribyl, B., *Oracle PL/SQL Programming*, 2nd Edition, O'Reilly, 1997

[2] Feuerstein, S., Dye, Ch., Beresniewicz, *J., Oracle Built-in Packages*, O'Reilly, 1998

[3] Feuerstein, S., *Advanced Oracle PL/SQL Programming with Packages*, O'Reilly, 1996

[4] Feuerstein, S., Odewahn, A., *Oracle PL/SQL Developer's Workbook*, O'Reilly, 2000

[5] Kyte, Thomas, *Effective Oracle by Design*, McGraw-Hill

[6] Lonely, K., Koch, G., *Oracle 9i – The Complete Reference*, McGraw-Hill 2002

[7] Trezzo, J., Brown, B., Niemiec, R., *Oracle PL/SQL Tips and Techniques*, McGraw-Hill, 1999

[8] Oracle on-line documentation at CERN
http://oracle-documentation.web.cern.ch/oracle-documentation/

[9] The Oracle PL/SQL CD Bookshelf on-line
http://cdbox.home.cern.ch/cdbox/GG/ORABOOKS/index.htm

[10] Ask Tom (Tom Kyte) http://asktom.oracle.com

End;

Thank you for your attention!

Zornitsa.Zaharieva@cern.ch