CSCI317 Database Performance Tuning

# Partitioning

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

# Partitioning

## Outline

# Partitioning ? What is it ?

Partitioning is a strategy for solving hard computationally complex problems

Partitioning is based on an idea of breaking large, and difficult problems into smaller ones and solving them one by one

Range partitioning divides data into smaller chunks such that some of them can be ignored during data processing

Partitioning of relational tables is transparent to user applications;

Partitioned relational table or index is processed by the applications as one table or one index

Physically a partitioned relational table or index consists of many smaller components

Partitioning uses a concept of partition key to determine which partition a row belongs to

Partition key may be based on range of values, a list of specific values, or a value of hash function

# Partitioning

## Outline

# Advantages of partitioning

Performance related observations

Partitioning increases availability of data

Partitioning eases administration of large relational tables

Partitioning improves performance of some queries

Partitioning may reduce contention on high volume online processing system by spreading out modifications over many separate partitions

Some partitions do not need to be considered for query processing

Parallel DML is possible on partitioned relational tables

Partitions can be used to increase concurrency by elimination of access conflicts, e.g. many partitioned indexes

Partitioning speeds up processing in data warehouse/decision-support systems

# Advantages of partitioning

## Independence of each partition increases availability

Partitioned relational table

```
CREATE TABLE EMP(
 enum   NUMBER(6)   NOT NULL,
 name   VARCHAR(30) NOT NULL,
  CONSTRAINT EMP_PKEY PRIMARY KEY(enum))
PARTITION BY HASH(enum)
(PARTITION EMP1 TABLESPACE TBS1,
 PARTITION EMP2 TABLESPACE TBS2);
```

INSERT into partitined relational table

```
INSERT INTO EMP VALUES( 1, 'James');
```

SELECT on a partition

```
SELECT *
FROM EMP PARTITION(EMP1);
```

# Advantages of partitioning

# Partitioning

## Outline

# Partitioning techniques

## Range partitioning

- Range partitioning uses ranges of data to determine the location of data; e.g. all transactions finalized in January

## Hash partitioning

- Hash partitioning uses hash function to determine the location of data; e.g. an attribute has a hash function applied to it

## List partitioning

- List partitioning uses discrete sets of values to determine the location of data; e.g. all rows with a value of attribute `STAT` equal to `'F'`,`'P'` should be located in the same partition

## Composite partitioning

- Composite partitioning is a combination of range, hash, and list partitioning; e.g. it allows to apply range partitioning to some data and later on within that range location of data is determined by hash or list partitioning

# Partitioning

## Outline

# Range partitioning

A sample range partitioned relational table `EMP`

Creating a range partitioned relational table

```
CREATE TABLE EMP(
 enum        NUMBER(6)       NOT NULL,
 name        VARCHAR(30)     NOT NULL,
 hired       DATE            NOT NULL,
      CONSTRAINT EMP_PKEY PRIMARY KEY(enum))
PARTITION BY RANGE(hired)
 (PARTITION P1 VALUES LESS THAN
      (TO_DATE('01-JAN-2005', 'DD-MON-YYYY'))
  PARTITION P2 VALUES LESS THAN
      (TO_DATE('01-JAN-2009', 'DD-MON-YYYY'))
);
```

SELECT statement accessing a range partitioned relational table

```
SELECT *
FROM EMP PARTITION(P2);
```

# Partitioning

## Outline

# List partitioning

A sample list partitioned relational table `EMP`

Creating a list partitioned relational table

```
CREATE TABLE EMP(
 enum        NUMBER(6)   NOT NULL,
 name        VARCHAR(30) NOT NULL,
 status      CHAR(1)     NOT NULL,
     CONSTRAINT EMP_PKEY PRIMARY KEY(enum))
PARTITION BY LIST(status)
 (PARTITION P1 VALUES ('S','M'),
  PARTITION P2 VALUES ('U'));
```

SELECT statement accessing a list partitioned relational table

```
SELECT *
FROM EMP
WHERE STATUS = 'U';
```

# Partitioning

## Outline

# Composite partitioning

A sample composite partitioned relational table `EMP`

Creating a composite partitioned relational table

```
CREATE TABLE EMP(
 enum     NUMBER(6)   NOT NULL,
 name     VARCHAR(30) NOT NULL,
 hired    DATE        NOT NULL,
        CONSTRAINT EMP_PKEY PRIMARY KEY(enum))
PARTITION BY RANGE(hired)
SUBPARTITION BY HASH(enum) SUBPARTITIONS 2
 (PARTITION P1 VALUES LESS THAN
      (TO_DATE('01-JAN-2005', 'DD-MON-YYYY'))
      (subpartition P1_1, subpartition P1_2)
PARTITION P2 VALUES LESS THAN
      (TO_DATE('01-JAN-2009', 'DD-MON-YYYY'))
      (subpartition P2_1, subpartition P2_2)
);
```

# Composite partitioning

A sample composite partitioned relational table `EMP`

Creating a composite partitioned relational table

```
CREATE TABLE EMP(
 enum     NUMBER(6)   NOT NULL,
 name     VARCHAR(30) NOT NULL,
 hired    DATE        NOT NULL,
        CONSTRAINT EMP_PKEY PRIMARY KEY(enum))
PARTITION BY RANGE(hired)
SUBPARTITION BY VALUES(name) SUBPARTITIONS 2
 (PARTITION P1 VALUES LESS THAN
      (TO_DATE('01-JAN-2005', 'DD-MON-YYYY'))
      (subpartition P1_1 values('James','Jane')
      subpartition P1_2 values('Kate','Ben'))
...
);
```

# References

Cookbook, How to create materialized views and partitioned relational tables ?

Lightstone S., Teorey T., Nadeau T., Physical Database Design Morgan Kaufman, 2007, chapters 6 and 7