Joins, Nested Loops (3.1)

```
SELECT *
FROM dept, emp;
```

```
>.SELECT STATEMENT
>...NESTED LOOPS
>.....TABLE ACCESS full dept
>.....TABLE ACCESS full emp
```

- Full Cartesian Product via Nested Loop Join (NLJ)
  - Init(RowSource1);
    While not eof(RowSource1)
    Loop Init(RowSource2);
    While not eof(RowSource2)
    Loop return(CurRec(RowSource1)+CurRec(RowSource2));
    NxtRec(RowSource2);
    End Loop;
    NxtRec(RowSource1);
    End Loop;

Two loops, nested

Joins, Sort Merge (3.2)

```
SELECT *
FROM emp, dept
WHERE emp.d# = dept.d#;
```

```
>.SELECT STATEMENT
>...MERGE JOIN
>.....SORT join
>.......TABLE ACCESS full emp
>.....SORT join
>.......TABLE ACCESS full dept
```

- Inner Join, no indexes: Sort Merge Join (SMJ)

```
Tmp1 := Sort(RowSource1,JoinColumn);
Tmp2 := Sort(RowSource2,JoinColumn);
Init(Tmp1); Init(Tmp2);
While Sync(Tmp1,Tmp2,JoinColumn)
Loop return(CurRec(Tmp1)+CurRec(Tmp2));
End Loop;
```

Sync advances pointer(s) to next match

Joins (3.3)

```
SELECT *
FROM emp, dept
WHERE emp.d# = dept.d#;

Emp(d#)
```

```
>.SELECT STATEMENT
>...NESTED LOOPS
>.....TABLE ACCESS full dept
>.....TABLE ACCESS by rowid emp
>.......INDEX range scan e_emp_fk
```

- Inner Join, only one side indexed
  - NLJ starts with full scan of non-indexed table
  - Per row retrieved use index to find matching rows
    - Within 2nd loop a (current) value for d# is available!
    - And used to perform a range scan

Joins (3.4)

```
SELECT *
FROM emp, dept
WHERE emp.d# = dept.d#

Emp(d#)
Unique Dept(d#)
```

```
>.SELECT STATEMENT
>...NESTED LOOPS
>.....TABLE ACCESS full dept
>.....TABLE ACCESS by rowid emp
>.......INDEX range scan e_emp_fk
Or,
>.SELECT STATEMENT
>...NESTED LOOPS
>.....TABLE ACCESS full emp
>.....TABLE ACCESS by rowid dept
>.......INDEX unique scan e_dept_pk
```

- Inner Join, both sides indexed
  - RBO: NLJ, start with FTS of last table in FROM-clause
  - CBO: NLJ, start with FTS of biggest table in FROM-clause
    - Best multi-block I/O benefit in FTS
    - More likely smaller table will be in buffer cache

Joins (3.5)

```
SELECT *
FROM emp, dept
WHERE emp.d# = dept.d#
AND dept.loc = 'DALLAS'

Emp(d#)
Unique Dept(d#)
```

```
>.SELECT STATEMENT
>...NESTED LOOPS
>.....TABLE ACCESS full dept
>.....TABLE ACCESS by rowid emp
>.......INDEX range scan e_emp_fk
```

- Inner Join with additional conditions
  - Nested Loops
  - Always starts with table thas has extra condition(s)

Hashing

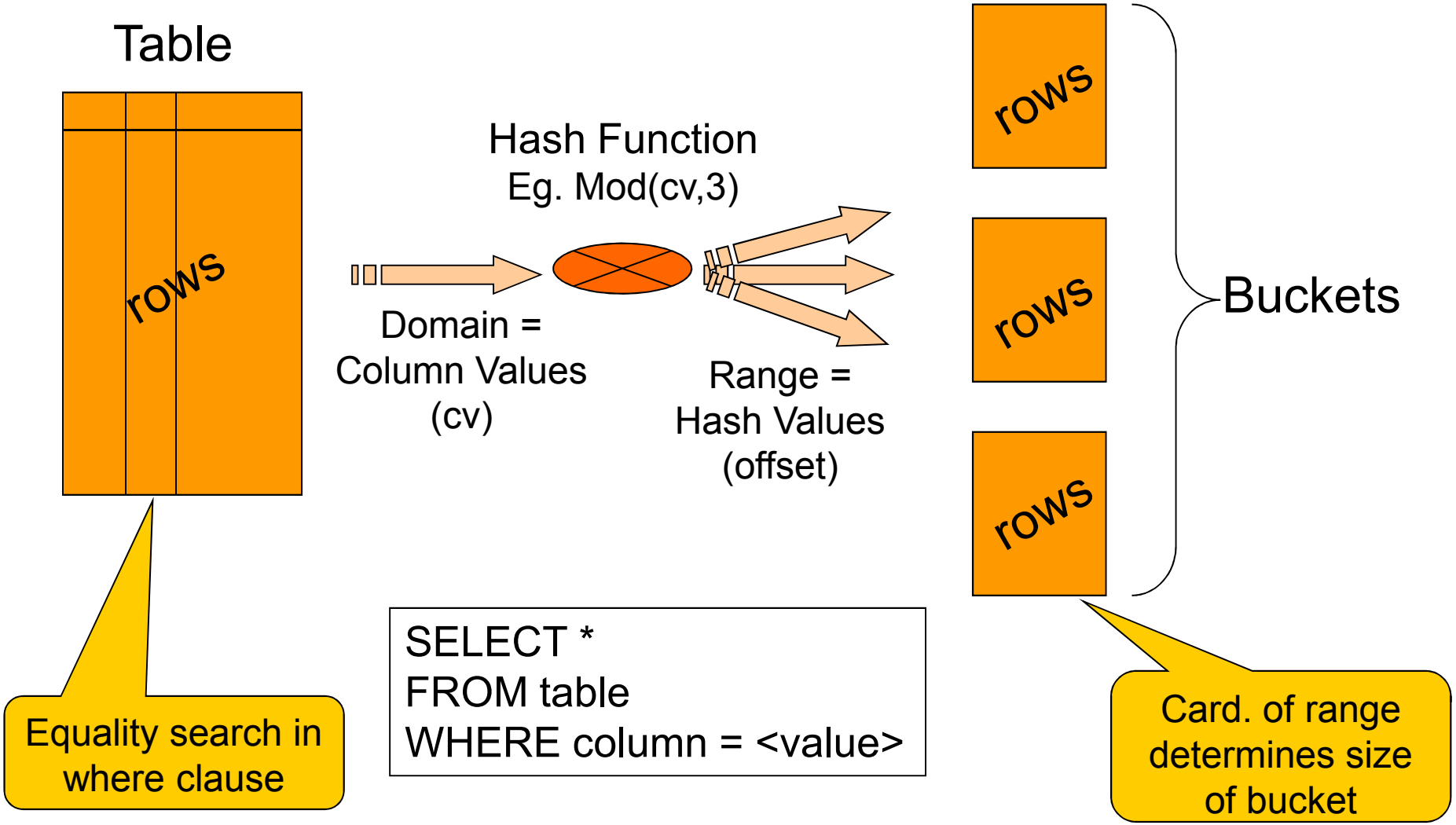
Table
rows
Hash Function
Eg. Mod(cv,3)
Domain =
Column Values
(cv)
Range =
Hash Values
(offset)
rows
rows
rows
Buckets
Equality search in where clause
SELECT *
FROM table
WHERE column = <value>
Card. of range determines size of bucket

Joins, Hash (3.6)

```
SELECT *
FROM dept, emp
WHERE dept.d# = emp.d#


Emp(d#), Unique Dept(d#)
```

```
>.SELECT STATEMENT
>...HASH JOIN
>.....TABLE ACCESS full dept
>.....TABLE ACCESS full emp
```

```
– Tmp1 := Hash(RowSource1,JoinColumn);      -- In memory
  Init(RowSource2);
  While not eof(RowSource2)
  Loop HashInit(Tmp1,JoinValue);              -- Locate bucket
          While not eof(Tmp1)
          Loop return(CurRec(RowSource2)+CurRec(Tmp1));
                  NxtHashRec(Tmp1,JoinValue);
          End Loop; NxtRec(RowSource2);
  End Loop;
```

Joins, Hash (3.6)

- Must be explicitely enabled via init.ora file:
  - Hash_Join_Enabled = True
  - Hash_Area_Size = <bytes>
- If hashed table does not fit in memory
  - 1st rowsource: temporary hash cluster is built
    - And written to disk (I/O's) in partitions
  - 2nd rowsource also converted using same hash-function
  - Per 'bucket' rows are matched and returned
    - One bucket must fit in memory, else very bad performance