

ORACLE®



# ORACLE®

## Rdb Enhancements

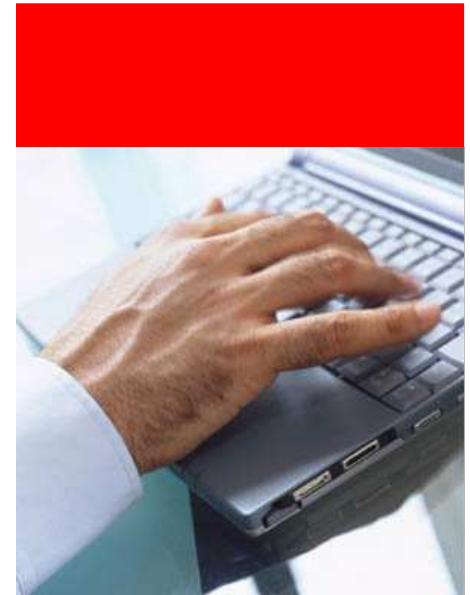
## Strings and Things

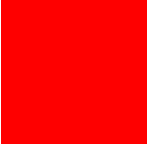
Ian Smith  
Oracle Rdb Product Architect



## Presentation Agenda

- Review V7.2 changes up to V7.2.5
- New functions and operators (7.2.5)
- Query rewrite
- Changes to LIKE
- Changes to CONCAT
- Final comments





## V7.2.5


- Significant release
- Contains new functionality
- Some internal re-structuring for performance



# New Syntax

## Some small improvements





## **SYSTIMESTAMP – system timestamp**

- Added for Oracle RDBMS compatibility
- Result is always TIMESTAMP
- Companion to SYSDATE
- SYSDATE functions like CURRENT\_TIMESTAMP
- SYSDATE and CURRENT\_TIMESTAMP morph based on setting of SET DEFAULT DATE FORMAT



## **SYS\_GUID – universally unique identifier**

- Global Unique ID
  - Same name as used in Oracle RDBMS (but different contents)
  - Based on OpenVMS SYS\$CREATE\_UID
  - Binary value CHAR(16) may include zero bytes so take care with C applications
  - Allows cluster wide unique id (possibly for PRIMARY KEY constraint or UNIQUE index)
- Can also specify GUID literals
- Support GUID typed domains
- Interactive SQL is GUID aware and displays formatted values

## GUID Examples

```
SQL> create domain GUID_DOMAIN char(16) character set -11;  
SQL show domain GUID_DOMAIN;  
GUID_DOMAIN          CHAR(16)  
GUID 16 Characters, 16 Octets
```

```
SQL> create table SAMPLE  
cont> (a int  
cont> ,b GUID_DOMAIN default  
cont> _guid'00000000-0000-0000-0000-000000000000');
```

```
SQL> select * from SAMPLE  
cont> where b =  
cont> _guid'3DBB657F-8513-11DF-9B74-0008029189E7' ;
```





## CONCAT\_WS

- Special form of CONCAT function
- \_WS means “with separator”
- First argument is used as a separator for all other arguments
- Implicit argument conversion to VARCHAR
- NULL values ignored

## CONCAT\_WS example (create CSV output)

```
SQL> select '' ||  
cont> CONCAT_WS ('', '' ,  
cont>         first_name, nvl(middle_initial,''), last_name) || ''  
cont> from employees  
cont> order by employee_id;
```

```
"Alvin    ", "A", "Toliver    "
```

```
"Terry    ", "D", "Smith      "
```

```
"Rick     ", "", "Dietrich  "
```

```
...
```

```
"Johanna  ", "P", "MacDonald  "
```

```
"James    ", "Q", "Herbener   "
```

```
100 rows selected
```



## DECLARE LOCAL TEMPORARY TABLE

- When this type of scratch table is defined **COMPRESSION IS ENABLED** always
- Sometimes the type of data isn't compressible or is very small
- Added **COMPRESSION** clause to allow disabling of compression to save CPU time
- For consistency also added to regular **CREATE TABLE** (previously required a **STORAGE MAP**, even for **TEMPORARY** tables)

## DECLARE TEMPORARY

```
SQL> declare local temporary table module.scratch0
cont>   (averages double precision)
cont>   compression is DISABLED
cont>   on commit PRESERVE rows
cont> ;
SQL>
SQL> insert into module.scratch0
cont>   select avg (char_length (a)) from module.scratch1;
1 row inserted
SQL>
SQL> select * from module.scratch0;
      AVERAGES
2.100000000000000000E+001
```



## Behind the curtain

**Changes you might not have noticed**





## Continual product tuning and development

- Nearly every point release includes changes aimed to improve underlying performance
  - Derived from research into customer problems
  - From internal benchmarking
  - Major development initiatives
- Usually release noted
- ...but sometimes you might miss them



## Efficient Execution

- Continual effort to eliminate “alignment faults” from our own software (SQL/Services, Interactive SQL, etc)
- Delivered native instruction compiler for Integrity (default in 7.2.3)
  - Continued refinement of generated code
  - Smaller code sequences
  - More overlapping execution
  - Increased use of zero source registers (R0)
  - Reduce alignment faults from generated queries
- Many changes positively affect Alpha systems also



## RDMS\$BIND\_WORK\_VM

- Used to tune “temporary relation” and “zig-zag” strategy
- Intermediate results are written to virtual memory up to this limit
- Then it will overflow to a temporary file (controlled by the logical RDMS\$BIND\_WORK\_FILE)
- Each occurrence in a query will use a separate buffer



## Zig-Zag tactic

Tables:

0 = EMPLOYEES

1 = DEPARTMENTS

Match (Left Outer Join) Q2

Outer loop

Match\_Key:0.EMPLOYEE\_ID

Get Retrieval by index of relation 0:EMPLOYEES

Index name EMP\_EMPLOYEE\_ID [0:0]

Inner loop (zig-zag)

Match\_Key:1.MANAGER\_ID

Index\_Key:MANAGER\_ID

Get Retrieval by index of relation 1:DEPARTMENTS

Index name DEP\_MANAGER\_ID [0:0]

## Temporary Relation

Tables:

0 = EMPLOYEES

1 = DEPARTMENTS

Conjunct: 1.MANAGER\_ID > '00180'

Match (Left Outer Join) Inner\_TTBL Q2

Outer loop

Match\_Key:0.EMPLOYEE\_ID

Get Retrieval by index of relation 0:EMPLOYEES

Index name EMP\_EMPLOYEE\_ID [0:0]

Inner loop

Match\_Key:1.MANAGER\_ID

Temporary relation

Sort: 1.MANAGER\_ID(a)

Conjunct: 1.MANAGER\_ID > '00180'

Get Retrieval by index of relation 1:DEPARTMENTS

Index name DEPARTMENTS\_INDEX [0:0]



## RDMS\$BIND\_WORK\_VM

- Restructured to use P2 space
- Frees up P0 space for user application
- Larger address space so we expanded from 10,000 bytes to 100,000 bytes
- This could mean that some queries that performed RMS I/O to the work file will now complete in virtual memory
- Better performance



## Sort restructuring

- Previously large SORT data structures were allocated in P0 space
- Many queries perform many SORT operations
  - ORDER BY
  - UNION DISTINCT
  - GROUP BY
  - DISTINCT
- Each “sort” requires own context and structures
- These data structures have now been moved to P2 space



## Sort restructuring

- Simple QSORT interface is used for small data sets
- Introduced in Rdb V7.2 to speed small sorts
- Controlled using two logical names
  - RDMS\$BIND\_MAX\_QSORT\_COUNT  
controls maximum number of rows
  - RDMS\$BIND\_MAX\_QSORT\_BUFFER  
controls total memory used to buffer sort rows
  - Used together to manage VM usage
- NEW: Moved buffers to P2 space
- Default threshold changed to allow larger sort sets



## Sort changes

- RDMS\$BIND\_MAX\_QSORT\_COUNT has been increased to **5,000** from 63
- RDMS\$BIND\_MAX\_QSORT\_BUFFER is no longer used
- Should allow more queries to avoid setup overhead for SORT32 interface

# Query Rewrite

## Internal query restructuring





## Query Rewrite

- Performed by the query compiler to produce simpler queries
- Easier to optimize
- Some current examples:
  - EXCEPT DISTINCT (aka MINUS)
  - INTERSECT DISTINCT
  - AND and OR sequences
  - EXISTS
  - RIGHT OUTER JOIN
  - IN clause
- Adding more in future major release



## EXISTS rewritten to COUNT <> 0

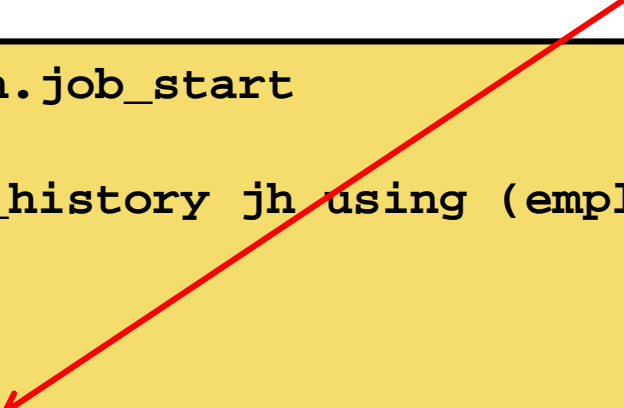
```
SQL> select last_name
cont> from employees
cont> where exists (select * from candidates
cont>                  where last_name = employees.last_name);
Tables:
  0 = EMPLOYEES
  1 = CANDIDATES
Cross block of 2 entries  Q1
Cross block entry 1
  Get      Retrieval sequentially of relation 0:EMPLOYEES
Cross block entry 2
  Conjunct: <agg0> <> 0
  Aggregate-F1: 0:COUNT-ANY (<subselect>) Q2
  Conjunct: 1.LAST_NAME = 0.LAST_NAME
  Get      Retrieval sequentially of relation 1:CANDIDATES
```

## MINUS rewritten as NOT EXISTS

```
SQL> select last_name from candidates
cont> minus select last_name from employees;
Tables:
  0 = CANDIDATES
  1 = EMPLOYEES
Reduce: 0.LAST_NAME
Sort: 0.LAST_NAME(a)
Merge of 1 entries  Q1
  Merge block entry 1  Q2
  Cross block of 2 entries  Q2
    Cross block entry 1
      Get Retrieval sequentially of relation 0:CANDIDATES
    Cross block entry 2
      Conjoinct: <agg0> = 0
      Aggregate-F1: 0:COUNT-ANY (<subselect>) Q3
      Conjoinct: (0.LAST_NAME = 1.LAST_NAME) OR
                  (MISSING (0.LAST_NAME) AND MISSING (1.LAST_NAME))
      Get Retrieval sequentially of relation 1:EMPLOYEES
```

## RIGHT OUTER rewritten as LEFT OUTER

```
SQL> select e.last_name, jh.job_start
cont> from employees e
cont> right outer join job_history jh using (employee_id);
Tables:
  0 = EMPLOYEES
  1 = JOB_HISTORY
Match      (Left Outer Join)  Q2
Outer loop
Match_Key:1.EMPLOYEE_ID
  Get      Retrieval by index of relation 1:JOB_HISTORY
           Index name  JH_EMPLOYEE_ID [0:0]
Inner loop      (zig-zag)
Match_Key:0.EMPLOYEE_ID
Index_Key:EMPLOYEE_ID
  Get      Retrieval by index of relation 0:EMPLOYEES
           Index name  EMP_EMPLOYEE_ID [0:0]
```



## IN clause rewritten to OR tree

```
SQL> select last_name from employees
cont> where first_name in ('Janet', 'Andrew');
Tables:
  0 = EMPLOYEES
Conjunct: (0.FIRST_NAME = 'Janet')
          OR (0.FIRST_NAME = 'Andrew')
Get      Retrieval sequentially of relation 0:EMPLOYEES
LAST_NAME
Kilpatrick
1 row selected
```



## Query rewrite - NEGATE

- New in V7.2.5
- Eliminate NEGATE operator in some cases
- Result is shorter code path which might be significant when many rows are processed
- Completely avoid NEGATE operation:
  - $-A + B$  transforms to  $B - A$
  - $A + -B$  transforms  $A - B$
  - $A - -B$  transforms  $A + B$



## Query Rewrite for string operators

- New in V7.2.5
- Implemented rewrite for LIKE, STARTING WITH and CONTAINING predicates
- Introduce SET FLAGS keyword REWRITE (or NOREWRITE) to change default setting
  - REWRITE(LIKE)
  - REWRITE(CONTAINING)
  - REWRITE(STARTING\_WITH)
- Allows Rdb engineering to test and compare semantics



# String Operators

**Any operator or function that  
use CHAR and VARCHAR  
types**





## String Operators

- Any operator that processes string values. For example:
  - Boolean operations
    - CONTAINING
    - STARTING WITH
    - LIKE
    - LIKE ... IGNORE CASE
  - Functions
    - CONCAT
    - SUBSTRING
    - POSITION
    - TRIM





## Changes to syntax

- New MATCHING operator
- LIKE operator
  - New ESCAPE clause supported with IGNORE CASE



## Changes to semantics

- LIKE ... IGNORE CASE
  - Support ESCAPE clause
  - Closer to ANSI and ISO SQL definition for LIKE
  - Allows use of columns and complex expressions for pattern
  - Can be used in a BEGIN/END compound statement



## Changes to behavior

- STARTING WITH and CONTAINING now handle zero length patterns differently
- LIKE and STARTING WITH handle exact match differently



## STARTING WITH and CONTAINING

- From time to time see queries that use these operators with zero length strings
- Really not a good selector for an index column and skews optimizer strategy
- Replace with length check to preserve NULL semantics
- Allows optimizer to make better choices

## STARTING WITH

- If the pattern string is the same size as the source expression then use EQUAL (=)
- Allows more efficient strategy creation

```
SQL> select last_name from employees
cont> where last_name starting with 'Smith' ;
Tables:
  0 = EMPLOYEES
Index only retrieval of relation 0:EMPLOYEES
  Index name  LAST_NAME_INDEX [1:1]
    Keys: 0.LAST_NAME = 'Smith'
  LAST_NAME
  Smith
  Smith
2 rows selected
```



# **MATCHING Operator**

## **Simple string patterns**





## MATCHING

- Originally introduced in RDO interface
- In older versions used by SQL for LIKE ... IGNORE CASE support
- VMS wildcards
  - % as matching a single character
  - \* as matching zero or more characters
- Case insensitive

# MATCHING

```
SQL> set flags 'strategy,detail(2)';
SQL> select first_name, last_name from employees
cont> where last_name matching 'smith*';
Tables:
  0 = EMPLOYEES
Leaf#01 FFirst 0:EMPLOYEES Card=100
  Bool: 0.LAST_NAME MATCHES 'smith*'
  BgrNdx1 LAST_NAME_INDEX [0:0] Fan=12
    Bool: 0.LAST_NAME MATCHES 'smith*'
FIRST_NAME    LAST_NAME
Terry         Smith
Roger         Smith
2 rows selected
SQL>
```



# LIKE ... IGNORE CASE Operator

**Case insensitive patterns**





## IGNORE CASE

- Originally used RDO style MATCHING operator
- Wildcards were different from SQL standard
- IGNORE CASE required pre-processing of string pattern (SQL wildcards to RDO style)
- Introduced several problems when pattern contained \* as data
- Not supported in compound statements
- No support for ESCAPE clause



## LIKE ... IGNORE CASE

- All new implementation
  - Based on existing LIKE implementation
  - Supports use of ESCAPE clause
  - Can be used with full range of pattern expressions
  - Supported in compound statements
- 
- SQL\$PRE and SQL\$MOD applications will require recompile to see new implementation
- 
- Note: diacritical marks no longer ignored

## LIKE ... IGNORE CASE

```
SQL> set flags 'strategy,detail(2)';
SQL> select last_name from employees
cont> where last_name like 'smith%' ignore case escape '/';
Tables:
  0 = EMPLOYEES
Conjunct: 0.LAST_NAME LIKE 'smith%' ESCAPE '/' IGNORE CASE
Index only retrieval of relation 0:EMPLOYEES
  Index name  FULL_NAME [0:0]
    Keys: 0.LAST_NAME LIKE 'smith%' ESCAPE '/' IGNORE CASE
LAST_NAME
Smith
Smith
2 rows selected
```



## LIKE ... IGNORE CASE

- If source doesn't support casing (for examples KANJI) then prior releases would generate an error
- Now SQL ignores clause, and issues compile time warning



## Changes to LIKE

## Visible and Hidden changes





## LIKE changes

- Focus was on lower CPU usage for LIKE
  - Optimized VM usage, and prevented release of local memory until query completes (instead of per row)
- Eliminated unnecessary runtime processing when we have literal LIKE pattern strings
- Reworked existing optimizations
- Added new query rewrite for LIKE operator



## LIKE optimizations

- *expr LIKE pattern* can sometimes be transformed

expr LIKE pattern  
AND expr STARTING WITH  
SUBSTRING(pattern FROM *n*)

- For example, a pattern such as ‘Sm%th’ becomes
- last\_name STARTING WITH ‘Sm’  
AND last\_name LIKE ‘Sm%th’
- The STARTING WITH allows partial index key match





## LIKE with STARTING addition

- Rdb V6.1 added pattern analysis to the generated query so that any leading prefix could be used to start the index lookup
- Not reflected in the displayed query strategy (until now)
  - See SET FLAGS ‘STRATEGY,DETAIL’
- Rdb now disables optimization if pattern is a string literal with leading wildcard
  - Saves runtime overhead

## Prior to V7.2.5 Example (this is V7.1)

```
SQL> select *
cont> from PERSON
cont> where last_name like '%Smith%';
Tables:
  0 = PERSON
Leaf#01 FFirst 0:PERSON Card=10
  Bool: 0.LAST_NAME LIKE '%Smith%'
  BgrNdx1 PERSON_NAME [1:1] Fan=10
    Keys: 0.LAST_NAME LIKE '%Smith%'
    Bool: 0.LAST_NAME LIKE '%Smith%'
```

Misleading  
notation

- Runtime code processed the LIKE pattern for every index scan.

## V7.2.5 Example

```
SQL> declare :ln varchar(10) = '%Smith%';
SQL> select *
cont> from PERSON
cont> where last_name like :ln;
Tables:
  0 = PERSON
Leaf#01 FFirst 0:PERSON Card=10
  Bool: 0.LAST_NAME LIKE <var0>
  BgrNdx1 PERSON_NAME [1:1] Fan=10
    Keys: 0.LAST_NAME LIKE <var0> (Starting With)
    Bool: 0.LAST_NAME LIKE <var0>
```

New  
annotation

- Host variable means we have no choices

## V7.2.5 Example

```
SQL> select *
cont> from PERSON
cont> where last_name like '%Smith%'
Tables:
  0 = PERSON
Leaf#01 FFIRST 0:PERSON Card=10
  Bool: 0.LAST_NAME LIKE '%Smith%'
  BgrNdx1 PERSON_NAME [0:0] Fan=10
    Keys: 0.LAST_NAME LIKE '%Smith%'
    Bool: 0.LAST_NAME LIKE '%Smith%'
```

Better  
indication

- Eliminate runtime overhead of pattern analysis - know for sure there is no prefix



## LIKE optimizations

- When the LIKE pattern is a literal with no wildcards we can often eliminate the LIKE operator completely

```
expr LIKE 'Smith'
```

- If the source expr is the same length, character set and the pattern contains no wildcards then it is transformed into

```
expr = 'Smith'
```



## Example

```
SQL> select *  
cont> from PERSON  
cont> where last_name like 'Smith  
' ;  
Tables:  
  0 = PERSON  
Leaf#01 FFIRST 0:PERSON Card=10  
  Bool: 0.LAST_NAME = 'Smith'  
  BgrNdx1 PERSON_NAME [1:1] Fan=10  
  Keys: 0.LAST_NAME = 'Smith'
```

## LIKE optimizations

- If the source is a VARCHAR with the same character set and the pattern contains no wildcards then it is transformed with a length check

```
SQL> select *
cont> from PERSON_UPDATE_LIST
cont> where last_name like 'Smith      ';
Tables:
      0 = PERSON_UPDATE_LIST
Leaf#01 FFirst 0:PERSON_UPDATE_LIST Card=10
      Bool: (0.LAST_NAME = 'Smith      ') AND
(OCTET_LENGTH (0.LAST_NAME) = 10)
      BgrNdx1 PERSON_UL_NAME [1:1] Fan=10
      Keys: 0.LAST_NAME = 'Smith      '
```



## LIKE optimizations

- When the LIKE pattern is a string literal we can often replace the LIKE operator with STARTING WITH

```
expr LIKE 'Sm%'
```

- Is equivalent to

```
expr STARTING WITH 'Sm'
```

- Allows optimizer to use partial index key for SORTED indices and eliminates the pattern matching



## V7.2.5 Example

```
SQL> select *  
cont> from PERSON  
cont> where last_name like 'Smith%';  
Tables:  
  0 = PERSON  
Leaf#01 FFirst 0:PERSON Card=10  
  Bool: 0.LAST_NAME STARTING WITH 'Smith'  
  BgrNdx1 PERSON_NAME [1:1] Fan=10  
  Keys: 0.LAST_NAME STARTING WITH  
  'Smith'
```

- Eliminate LIKE when possible
- No runtime pattern matching required

# CONCAT Operator

**Concatenate compatible  
strings**





## Concatenate

- Defined by the SQL Database Language Standard
- Uses the binary operator ||
- Part of SQL since first Rdb versions
- Can also use CONCAT (a, b) function
- Rules: if either a or b is NULL then concatenation will result in a NULL result



## Problem to be solved

- Each binary operator will compute and return an intermediate result
- Each subsequent intermediate result requires more memory – especially for multiple CONCAT usage



## Example

```
SQL> select employee_id, birthday  
cont> from employees  
cont> where (first_name || middle_initial || last_name)  
cont>         = 'Alvin      AToliver      ';
```

- FIRST\_NAME is CHAR(14)
- MIDDLE\_INITIAL is CHAR(1)
- LAST\_NAME is CHAR(20)
- || intermediate result is  $14 + 1 = 15$
- || intermediate result is  $15 + 20 = 35$
- Total overhead is 50 bytes



## Problem to be solved

- Observed applications that concatenate 50 or more strings (columns and literals)
- So intermediate results were tainting query execution
- Note that intermediate results require memory, CPU to copy the results, and so on



## Oracle RDBMS Semantics

- NULL Semantics:
  - Oracle CONCAT treats NULL values as zero length strings
  - If CONCAT results in a zero length result then it is considered NULL
- Enable these semantics using:  
`set dialect 'oracle level2' ;`
- Used to select different semantics for correct behavior of OCI (Oracle Call Interface) applications
- Level2 = SQL99 plus Oracle RDBMS specific semantics



## Semantics

- In V7.0 through V7.2 (early releases) used hidden CASE expressions to get the desired semantics
- **select**  
    **case first\_name**  
        **when is null then ‘ ’ else first\_name end ||**  
    **... ||**  
    **case last\_name**  
        **when is null then ‘ ’ else last\_name end**  
**from employees;**
- Adds considerable overhead at runtime
  - Can see this in the following example



## V7.0, Oracle LEVEL1 dialect

```
SQL> select * from employees
cont> where middle_initial = trim (last_name || first_name);
Tables:
  0 = EMPLOYEES
Conjunct: 0.MIDDLE_INITIAL = CASE (WHEN (CHAR_LENGTH (TRIM (BOTH ' ' FROM (
      CASE (WHEN MISSING (0.LAST_NAME) THEN ' ' ELSE 0.LAST_NAME) || CASE (
      WHEN MISSING (0.FIRST_NAME) THEN ' ' ELSE 0.FIRST_NAME)))) = 0) THEN
      NULL ELSE TRIM (BOTH ' ' FROM (CASE (WHEN MISSING (0.LAST_NAME) THEN
      ' ' ELSE 0.LAST_NAME) || CASE (WHEN MISSING (0.FIRST_NAME) THEN ' '
      ELSE 0.FIRST_NAME))))
Get      Retrieval sequentially of relation 0:EMPLOYEES
0 rows selected
```



## We had a problem to solve

- Normal CONCAT consumed a lot of memory
- But adding ORACLE DIALECT caused the conditional expressions to also introduce additional memory requirements and CPU overhead
- Full redesign was released in 7.2.1

## V7.1, default dialect

```
SQL> set flags 'strategy,detail(2)';
SQL>
SQL> select employee_id, birthday
cont> from employees
cont> where (first_name || middle_initial || last_name)
cont>        = 'Alvin      AToliver      ';
Tables:
  0 = EMPLOYEES
Conjunct: (0.FIRST_NAME || 0.MIDDLE_INITIAL || 0.LAST_NAME) =
          'Alvin      AToliver      '
Get      Retrieval by index of relation 0:EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [0:0]
EMPLOYEE_ID  BIRTHDAY
00164        28-Mar-1947
1 row selected
```

## V7.1, Oracle dialect, case used to implement semantics

```
SQL> set dialect 'oracle level2';
SQL> select employee_id, birthday
cont> from employees
cont> where (first_name || middle_initial || last_name)
cont>      = 'Alvin      AToliver      ';
Tables:
      0 = EMPLOYEES
Conjunct: (CASE (WHEN MISSING (0.FIRST_NAME)
                THEN ' ' ELSE 0.FIRST_NAME) ||
          CASE (WHEN MISSING (0.MIDDLE_INITIAL)
                THEN ' ' ELSE 0.MIDDLE_INITIAL)
          || CASE (WHEN MISSING (0.LAST_NAME)
                THEN ' ' ELSE 0.LAST_NAME)) =
          'Alvin      AToliver      '
Get      Retrieval by index of relation 0:EMPLOYEES
      Index name  EMP_EMPLOYEE_ID [0:0]
EMPLOYEE_ID      BIRTHDAY
00164            28-Mar-1947
1 row selected
```



## New in V7.2.1 and later

- Rdb re-implemented the CONCAT operator
- Was a binary (two valued) function – CONCAT(a,b)
- Now is an n-ary function (up to 65535 values)
- New implementation is more efficient and uses less virtual memory compared with old implementation
- No hidden case expressions
- Supports the Oracle RDBMS semantics

## V7.2, default dialect, new function used

```
SQL> set flags 'strategy,detail(2)';
SQL>
SQL> select employee_id, birthday
cont> from employees
cont> where (first_name || middle_initial || last_name)
cont>      = 'Alvin      AToliver      ';
Tables:
  0 = EMPLOYEES
Conjunct: CONCAT (0.FIRST_NAME, 0.MIDDLE_INITIAL,
                  0.LAST_NAME) =
          'Alvin      AToliver      '
Get      Retrieval by index of relation 0:EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [0:0]
EMPLOYEE_ID  BIRTHDAY
00164        28-Mar-1947
1 row selected
```



## Further optimizations - query rewrite

- We also know that we can rewrite such CONCAT queries to expose the column references to the optimizer
- Only makes sense if there exists an index to use
- Define index on LAST\_NAME and FIRST\_NAME columns

```
SQL> create index SORTED_NAMES  
cont> on EMPLOYEES (last_name, first_name)  
cont> type is sorted ranked;
```



## CONCAT

- Follows simple rule that:

CONCAT (a, b) = expr

a = SUBSTRING (expr FROM 1 to LENGTH(a))  
AND b = SUBSTRING (expr FROM LENGTH(a)+1)

- The columns a and b are now exposed to the optimizer for index retrieval



## V7.2, default dialect, after query rewrite, concatenation removed - index used

```
SQL> select employee_id, birthday
cont> from employees
cont> where (first_name || middle_initial || last_name)
cont>      = 'Alvin      AToliver      ';
Tables:
  0 = EMPLOYEES
Leaf#01 FFirst 0:EMPLOYEES Card=100
  Bool: (0.LAST_NAME = SUBSTRING ('Alvin  AToliver  ' FROM 11)) AND (
        0.MIDDLE_INITIAL = SUBSTRING ('Alvin  AToliver  ' FROM 10 FOR 1)
        ) AND (0.FIRST_NAME = SUBSTRING ('Alvin  AToliver  ' FROM 0 FOR
        10))
  BgrNdx1 SORTED_NAMES [2:2] Fan=9
    Keys: (0.LAST_NAME = SUBSTRING ('Alvin  AToliver  ' FROM 11)) AND (
          0.FIRST_NAME = SUBSTRING ('Alvin  AToliver  ' FROM 0 FOR 10))
EMPLOYEE_ID  BIRTHDAY
00164        28-Mar-1947
1 row selected
```



## Concatenation restrictions

- For Oracle dialect any columns that are NULL are ignored (treated as zero length)
- SUBSTRING doesn't reproduce the same NULL semantics
- Therefore, can not apply this rewrite to Oracle dialect queries



## Final Comments

**Much more to say...**





## What will we talk about next time?

- We discussed
  - New functions and syntax
  - Query Rewrite
- Did not get to talk about other new features from recent Rdb releases
  - DEFAULT PROFILE statement
  - PROFILE LIMIT clauses
  - New SQL SET statements



## Documentation

- More information on new and changed features maintained in the RDB\_NEWFEATURES\_72XX document (see SYS\$HELP)
- Revised versions of *SQL Reference Manual* and the *RMU Reference Manual* for V7.2.5 will be issued soon
- Currently undergoing documentation review
- All DOC related bug reports for these manuals have been closed



# Questions? & Answers!

**SOFTWARE. HARDWARE. COMPLETE.**

ORACLE®