

# FIREBIRD 2.1

## What's new

*Vladyslav Khorsun*  
hvlad@users.sourceforge.net

# FIREBIRD 2.1

---

- **BETTER ADMINISTRATION**

*Monitoring and Windows trusted authentication*

- **IMPROVED PERFORMANCE**

*Less roundtrips, faster large sorts and other...*

- **SQL LANGUAGE ENHANCEMENTS**

*GTT, CTE, Database Triggers and more...*

- **BUGS FIXED**

*Many bugs was fixed*



# ADMINISTRATION

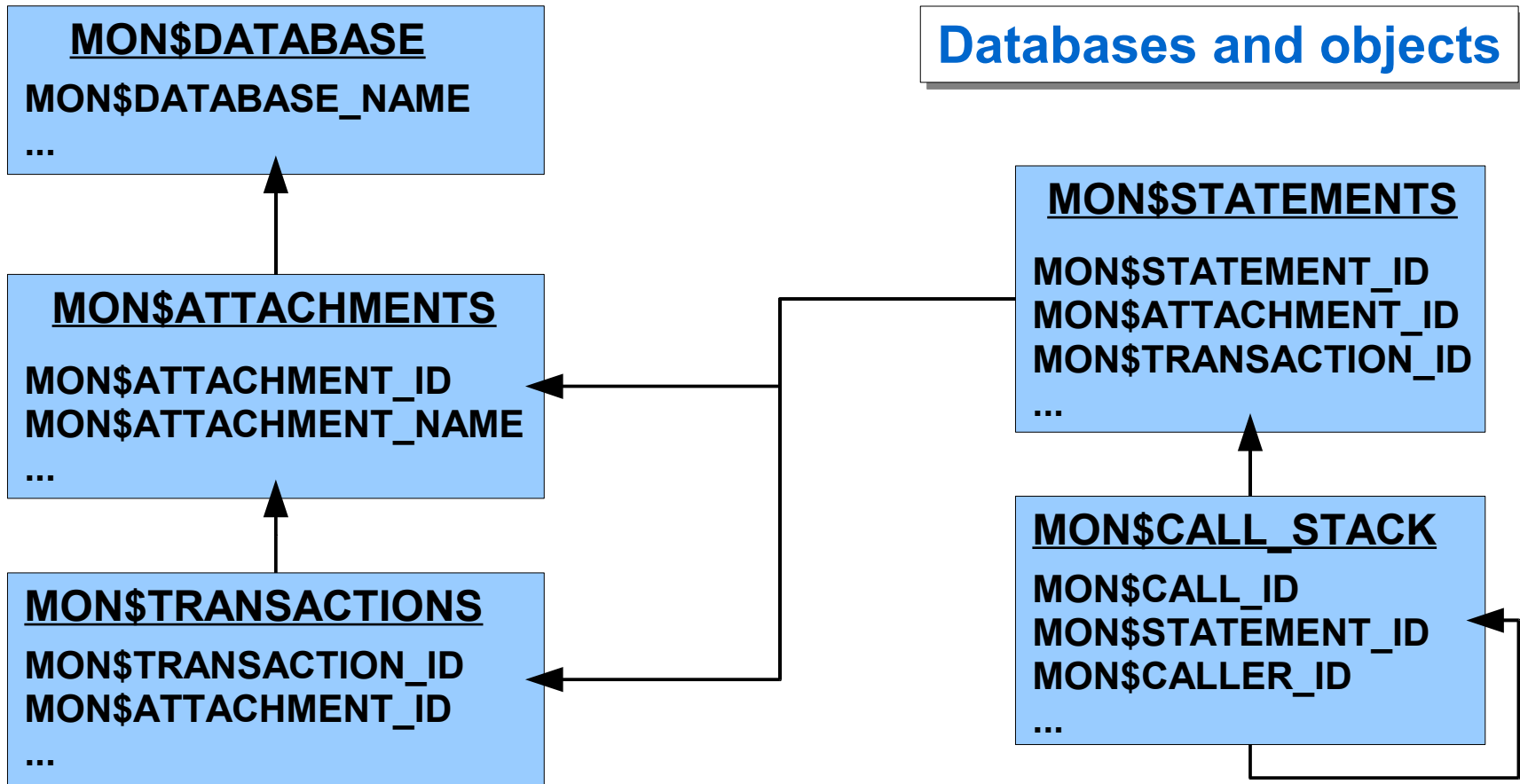
---

- **Monitoring tables**
  - **Activity snapshot retained till the end of transaction**
  - **Works with both Super and Classic Server**
  - **Regular users restricted to see own attachment**
  - **Kill long running query**

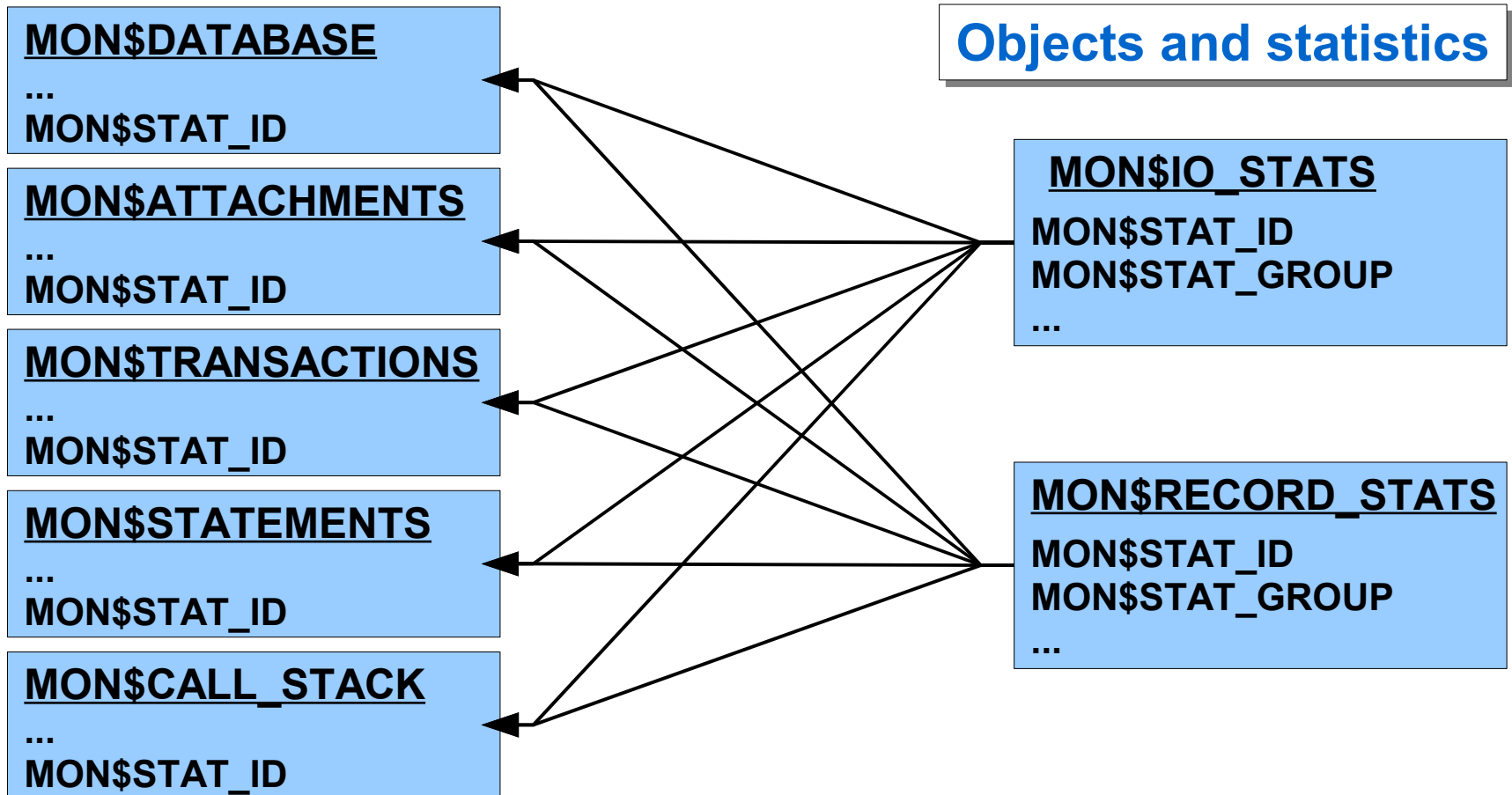


# MONITORING TABLES

## Databases and objects



# MONITORING TABLES



# ADMINISTRATION

---

- **Trusted Windows authentication**
  - Used built-in Windows security
  - New **SYSTEM\_USER** context variable (?)
  - New **Authentication** parameter in firebird.conf :
    - Native
    - Trusted
    - Mixed
  - New DPB tag **isc\_dpb\_trusted\_auth**
  - New command line switch **-trusted** in utilities



# PERFORMANCE

---

- Network protocol improvements
- Flush of large page cache
- Huge on-disk sorts
- External tables read\write performance



# NETWORK PROTOCOL IMPROVED

---

- **Less roundtrips and traffic saving**
  - **Some type of packets are delayed**
    - *(require both client and server v2.1)*
  - **Cached statement information**
    - *(client v2.1 with any server version)*
  - **No trailing zeros in information response**
    - *(any client with server v2.1)*
- **Full backward compatibility**



# SQL LANGUAGE

---

- **DATABASE TRIGGERS**

*Attachment and transaction level triggers*

- **GLOBAL TEMPORARY TABLE**

*New kind of tables*

- **COMMON TABLE EXPRESSIONS**

*Complex and recursive queries ? It is simple !*

- **DOMAINS EVERYWHERE**

*Domains in procedures, triggers and CAST*

- **INSERT OR UPDATE, MERGE**

*Insert or update ? Let Firebird decide ;)*

- **A LOT OF NEW BUILT-IN FUNCTIONS**



# DATABASE TRIGGERS

---

- Database-wide triggers fired on :
  - CONNECT
  - DISCONNECT
  - TRANSACTION START
  - TRANSACTION COMMIT
  - TRANSACTION ROLLBACK
- Only SYSDBA or database owner can :
  - define database triggers
  - switch it off for new connection by :
    - new **isc\_dpb\_no\_db\_triggers** tag
    - new **-no\_dbtriggers** switch in utilities



# DATABASE TRIGGERS

## Example of ON CONNECT trigger

```
isql temp.fdb -user SYSDBA -pass masterkey
Database: temp.fdb, User: SYSDBA
SQL> SET TERM ^ ;
SQL> CREATE EXCEPTION EX_CONNECT 'Forbidden !' ^
SQL> CREATE OR ALTER TRIGGER TRG_CONN ON CONNECT
CON> AS
CON> BEGIN
CON>   IF (<bad user>)
CON>   THEN EXCEPTION EX_CONNECT USER || ' not allowed !';
CON> END ^
SQL> EXIT ^
```

```
isql temp.fdb -user BAD_USER -pass ...
Statement failed, SQLCODE = -836
exception 217
-EX_CONNECT
-BAD_USER not allowed !
-At trigger 'TRG_CONN' line: 5, col: 3
Use CONNECT or CREATE DATABASE to specify a database
SQL> EXIT;
```



# DATABASE TRIGGERS

---

## Example of ON CONNECT trigger

**Help ! My ON CONNECT trigger deny all users and even SYSDBA ! Nobody can log in anymore :(**

**What can i do ???**

**Don't panic :) Use -no\_dbtriggers switch in utilities :**

```
isql temp.fdb -user SYSDBA -pass masterkey -nodbtriggers
Database: temp.fdb, User: SYSDBA
SQL> ALTER TRIGGER TRG_CONN INACTIVE;
SQL> EXIT;
```



# GLOBAL TEMPORARY TABLES

---

- **Standard properties**
  - Common metadata and private data
  - Indices, triggers, uniqueness check and referential integrity
  - Lifetime : attachment or transaction
  - DELETE triggers are not fired on cleanup
- **Firebird's specific**
  - No problem with garbage collection
  - Storage is in temporary files, forced writes off
  - Instant cleanup



# GLOBAL TEMPORARY TABLES

---

- Syntax

- CREATE **GLOBAL TEMPORARY** TABLE

...

**[ON COMMIT <DELETE | PRESERVE> ROWS]**

- ON COMMIT **DELETE** ROWS

*data lifetime - transaction*

- ON COMMIT **PRESERVE** ROWS

*data lifetime - attachment*

- By default - **DELETE** ROWS



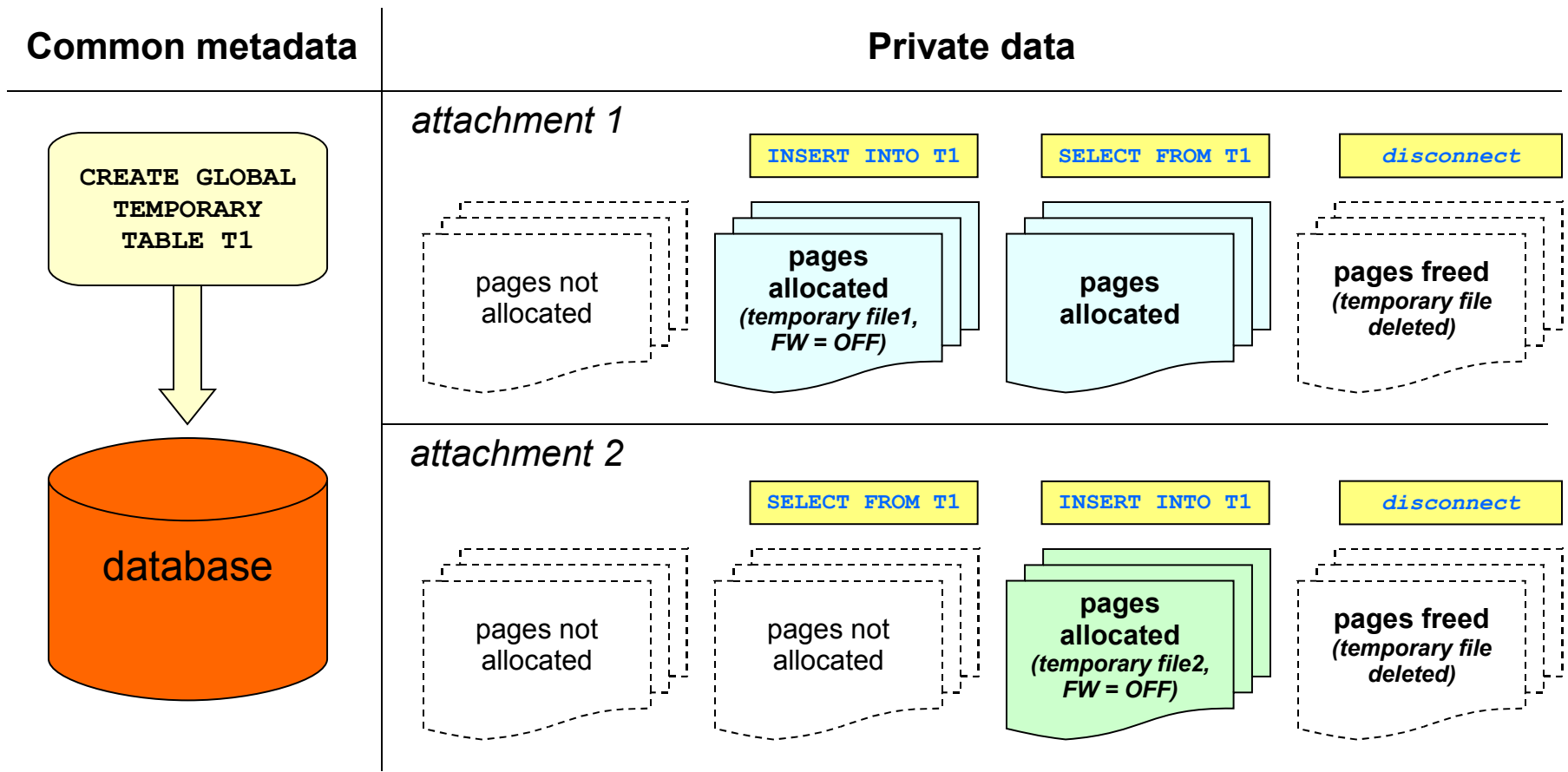
# GLOBAL TEMPORARY TABLES

## Referential integrity rules

master detail	<b>persistent</b>	<b>on commit preserve rows</b>	<b>on commit delete rows</b>
<b>persistent</b>	✓		
<b>on commit preserve rows</b>		✓	
<b>on commit delete rows</b>		✓	✓



# GLOBAL TEMPORARY TABLES



# COMMON TABLE EXPRESSIONS

---

- Syntax

- **WITH [RECURSIVE]** *-- new keywords*
  - **CTE\_A** *-- first table expression's name*  
[(a1, a2, ...)] *-- fields aliases, optional*  
AS ( SELECT ... ), *-- table expression's definition*
  - **CTE\_B** *-- second table expression*  
[(b1, b2, ...)]  
AS ( SELECT ... ),
  - ...
- **SELECT ...** *-- main query, used both*  
FROM **CTE\_A, CTE\_B,** *-- table expressions*  
TAB1, TAB2 *-- and regular tables*  
WHERE ...



# COMMON TABLE EXPRESSIONS

---

## Rules of simple (non recursive) table expressions

- Several table expressions can be defined at one query
- Any SELECT's clause can be used in table expressions
- Table expressions can reference each other
- Table expressions can be used within any part of main query or another table expression
- The same table expression can be used several times in main query



# COMMON TABLE EXPRESSIONS

---

## Rules of simple (non recursive) table expressions

- Table expressions can be used in INSERT, UPDATE and DELETE statements (as subqueries of course)
- Table expressions can be used in procedure language also :

*for with ... select ... into ...*

- WITH statements can not be nested
- References between expressions should not have a loops



# COMMON TABLE EXPRESSIONS

## Example of simple (non recursive) table expressions

```
WITH
DEPT_YEAR_BUDGET AS (
  SELECT FISCAL_YEAR, DEPT_NO, SUM(PROJECTED_BUDGET) AS BUDGET
  FROM PROJ_DEPT_BUDGET
  GROUP BY FISCAL_YEAR, DEPT_NO
)
SELECT D.DEPT_NO, D.DEPARTMENT,
       B_1993.BUDGET AS B_1993, B_1994.BUDGET AS B_1994,
       B_1995.BUDGET AS B_1995, B_1996.BUDGET AS B_1996
FROM DEPARTMENT D
  LEFT JOIN DEPT_YEAR_BUDGET B_1993
ON D.DEPT_NO = B_1993.DEPT_NO AND B_1993.FISCAL_YEAR = 1993
  LEFT JOIN DEPT_YEAR_BUDGET B_1994
ON D.DEPT_NO = B_1994.DEPT_NO AND B_1994.FISCAL_YEAR = 1994
  LEFT JOIN DEPT_YEAR_BUDGET B_1995
ON D.DEPT_NO = B_1995.DEPT_NO AND B_1995.FISCAL_YEAR = 1995
  LEFT JOIN DEPT_YEAR_BUDGET B_1996
ON D.DEPT_NO = B_1996.DEPT_NO AND B_1996.FISCAL_YEAR = 1996

WHERE EXISTS (SELECT * FROM PROJ_DEPT_BUDGET B WHERE D.DEPT_NO = B.DEPT_NO)
```



# COMMON TABLE EXPRESSIONS

Example of simple (non recursive) table expressions

DEPT_NO	DEPARTMENT	B_1993	B_1994	B_1995	B_1996
0	Corporate Headquarters		100 000.00		
100	Sales and Marketing		1 500 000.00	3 500 000.00	150 000.00
621	Software Development	20 000.00	2 340 000.00	900 000.00	
622	Quality Assurance		560 000.00		
623	Customer Support		80 000.00	1 200 000.00	
670	Consumer Electronics Div.		20 000.00		
671	Research and Development		461 000.00		
672	Customer Services		100 000.00	800 000.00	
110	Pacific Rim Headquarters		200 000.00	1 200 000.00	



# COMMON TABLE EXPRESSIONS

---

## Recursive table expressions

- **WITH RECURSIVE**      *-- mandatory keyword **RECURSIVE***
  - CTE\_R AS (      *-- recursive table expression*
    - SELECT ...      *-- non recursive query (anchor member)*  
FROM T1
    - UNION ALL      *-- mandatory UNION ALL*
    - SELECT ...      *-- recursive query*  
FROM CTE\_R      *-- recursive reference*
  - )
  - ...      *-- another table expressions*
  - SELECT ...      *-- main query*  
FROM CTE\_R ...  
WHERE ...



# COMMON TABLE EXPRESSIONS

---

## Recursive table expressions

- Recursive CTE have reference to itself
- Recursive CTE is an UNION of recursive and non-recursive members
- At least one non-recursive member (anchor) must be present
- Non-recursive members are placed first in UNION
- Recursive members are separated from an anchor members and from each other with UNION ALL



# COMMON TABLE EXPRESSIONS

---

## Recursive table expressions

- References between CTE's still should not have a loops
- Aggregates (DISTINCT, GROUP BY, HAVING) and aggregate functions (SUM, COUNT, MAX etc) are not allowed in recursive members
- Recursive member can have only one reference to itself and only in FROM clause
- Recursive reference can not participate in outer joins



# COMMON TABLE EXPRESSIONS

## Examples of recursive table expressions

**WITH RECURSIVE**

```
R_TREE (ID, LEVEL, PATH) AS
(
  SELECT ID, 0, CAST(ID AS VARCHAR(255))
    FROM TREE
   WHERE PARENT_ID IS NULL

  UNION ALL

  SELECT TREE.ID, R_TREE.LEVEL + 1,
         R_TREE.PATH || '\'.' ||
         CAST(TREE.ID AS VARCHAR(8))
    FROM TREE, R_TREE
   WHERE TREE.PARENT_ID = R_TREE.ID
        AND R_TREE.LEVEL < 10
)

SELECT TREE.*, R_TREE.LEVEL, R_TREE.PATH
  FROM TREE, R_TREE
 WHERE TREE.ID = R_TREE.ID
```



# COMMON TABLE EXPRESSIONS

## Examples of recursive table expressions

```
CREATE TABLE PEOPLES
(
  PEOPLE_ID INT NOT NULL PRIMARY KEY,
  NAME      VARCHAR(255) ,
  FATHER    INT REFERENCES PEOPLES,
  MOTHER    INT REFERENCES PEOPLES
)
```

```
WITH RECURSIVE
FAMILY AS
(
  SELECT PEOPLE, FATHER, MOTHER
  FROM PEOPLES
  WHERE NAME = :CHILD

  UNION ALL

  SELECT F.PEOPLE_ID, F.FATHER, F.MOTHER
  FROM PEOPLES F, FAMILY
  WHERE F.PEOPLE_ID = FAMILY.FATHER

  UNION ALL

  SELECT M.PEOPLE_ID, M.FATHER, M.MOTHER
  FROM PEOPLES M, FAMILY
  WHERE M.PEOPLE_ID = FAMILY.MOTHER
)

SELECT * FROM FAMILY
```



# COMMON TABLE EXPRESSIONS

## Examples of recursive table expressions

WITH RECURSIVE

```
DEPT_YEAR_BUDGET AS
(
  SELECT FISCAL_YEAR, DEPT_NO,
         SUM(PROJECTED_BUDGET) AS BUDGET
  FROM PROJ_DEPT_BUDGET
  GROUP BY FISCAL_YEAR, DEPT_NO
),
DEPT_TREE AS
(
  SELECT DEPT_NO, HEAD_DEPT, DEPARTMENT,
         CAST(' ' AS VARCHAR(255)) AS INDENT
  FROM DEPARTMENT
  WHERE HEAD_DEPT IS NULL

  UNION ALL

  SELECT D.DEPT_NO, D.HEAD_DEPT, D.DEPARTMENT,
         H.INDENT || ' '
  FROM DEPARTMENT D JOIN DEPT_TREE H
  ON D.HEAD_DEPT = H.DEPT_NO
)
```

```
SELECT D.DEPT_NO,
       D.INDENT || D.DEPARTMENT AS DEPARTMENT,
       B_1993.BUDGET AS B_1993, B_1994.BUDGET AS B_1994,
       B_1995.BUDGET AS B_1995, B_1996.BUDGET AS B_1996

FROM DEPT_TREE D
  LEFT JOIN DEPT_YEAR_BUDGET B_1993
  ON D.DEPT_NO = B_1993.DEPT_NO
 AND B_1993.FISCAL_YEAR = 1993

  LEFT JOIN DEPT_YEAR_BUDGET B_1994
  ON D.DEPT_NO = B_1994.DEPT_NO
 AND B_1994.FISCAL_YEAR = 1994

  LEFT JOIN DEPT_YEAR_BUDGET B_1995
  ON D.DEPT_NO = B_1995.DEPT_NO
 AND B_1995.FISCAL_YEAR = 1995

  LEFT JOIN DEPT_YEAR_BUDGET B_1996
  ON D.DEPT_NO = B_1996.DEPT_NO
 AND B_1996.FISCAL_YEAR = 1996
```



# COMMON TABLE EXPRESSIONS

## Examples of recursive table expressions

DEPT_NO	DEPARTMENT	B_1993	B_1994	B_1995	B_1996
0	Corporate Headquarters		100 000.00		
100	Sales and Marketing		1 500 000.00	3 500 000.00	150 000.00
180	Marketing				
130	Field Office: East Coast				
140	Field Office: Canada				
110	Pacific Rim Headquarters		200 000.00	1 200 000.00	
115	Field Office: Japan				
116	Field Office: Singapore				
120	European Headquarters				
121	Field Office: Switzerland				
123	Field Office: France				
125	Field Office: Italy				
600	Engineering				
620	Software Products Div.				
621	Software Development	20 000.00	2 340 000.00	900 000.00	
622	Quality Assurance		560 000.00		
623	Customer Support		80 000.00	1 200 000.00	
670	Consumer Electronics Div.		20 000.00		
671	Research and Development		461 000.00		
672	Customer Services		100 000.00	800 000.00	
900	Finance				



# DOMAINS EVERYWHERE

---

- **Syntax**

```
data_type ::= <builtin_data_type>
           | <domain_name>
           | TYPE OF <domain_name>
```

- **Usage**

- **Domains in input and output parameters**

- CREATE PROCEDURE MY\_PROC (IN\_PARAM [**TYPE OF**] DOMAIN\_A)  
 RETURNS (OUT\_PARAM [**TYPE OF**] DOMAIN\_B)

- **Domains in variable declaration**

- DECLARE VARIABLE VAR1 [**TYPE OF**] DOMAIN\_A

- **Domains in CAST statement**

- OUT\_PARAM = CAST (VAR1 AS [**TYPE OF**] DOMAIN\_B)



# DOMAINS EVERYWHERE

---

- Using plain <domain\_name> data type inherits domain constraints (CHECK, NOT NULL) and default value
- Using new keyword **TYPE OF** only data type from domain definition is inherited.
- Engine tracked dependency between domain and stored procedure or trigger where its used and not allows to drop such domains
- When domain altered engine don't recompile dependent objects
- Dependent objects will take up new domain definition only after reload into metadata cache



# DOMAINS EVERYWHERE

## Example : domain's DEFAULT value

```
SQL> SET TERM ^ ;
SQL>
SQL> CREATE DOMAIN COLOUR VARCHAR(8)
CON>   DEFAULT 'RED'
CON>   CHECK (VALUE IN ('RED', 'GREEN', 'BLUE'))^
SQL>
SQL> EXECUTE BLOCK
CON>   RETURNS (C1 COLOUR, C2 TYPE OF COLOUR)
CON> AS
CON> BEGIN
CON>   SUSPEND;
CON> END ^
```

C1	C2
=====	=====
RED	<null>



# DOMAINS EVERYWHERE

## Example : domain's CHECK CONSTRAINT

```
SQL> CREATE OR ALTER PROCEDURE COLOUR_USING
CON> AS
CON> DECLARE C1 COLOUR;
CON> DECLARE C2 TYPE OF COLOUR;
CON> BEGIN
CON>   C2 = 'BLACK';
CON>   C1 = 'WHITE';
CON> END ^
SQL>
SQL> EXECUTE PROCEDURE COLOUR_USING ^
Statement failed, SQLCODE = -625
validation error for variable C1, value "WHITE"
-At procedure 'COLOUR_USING' line: 7, col: 3
SQL>
```



# UPDATE OR INSERT

## • Syntax

- **UPDATE OR INSERT** INTO <table> [(col1, ..., colN)]  
VALUES (val1, ..., valN)
- [**MATCHING** (mc1, ..., mcM)] -- new keyword **MATCHING**, optional
- [**RETURNING** (ret1, ..., retR)] -- optional **RETURNING ... INTO**  
[**INTO** :var1, ..., :varR] -- clause (introduced in FB 2.0)

*UPDATE OR INSERT statement is close equivalent of construction below but easier to write and understand*

```
UPDATE <table>
  SET col1 = val1, ... colN = valN
  WHERE mc1 = val1 AND ... AND mcM = valM);
IF (ROWCOUNT = 0)
THEN INSERT INTO <table> [(col1, ..., colN)]
  VALUES (val1, ..., valN)
```



# UPDATE OR INSERT

---

## UPDATE OR INSERT rules

- If optional clause **MATCHING** is omitted then primary key is used to match table rows against replaced values
- If **RETURNING** clause is used then no more than one row must correspond to **MATCHING** criteria
- “**IF (ROWCOUNT = 0) ...**” construction can be used in PSQL only while **UPDATE OR INSERT** can be used directly as it is usual SQL statement
- User must have both **INSERT** and **UPDATE** privileges on table
- If row was inserted then **INSERT** trigger fired else fired **UPDATE** trigger



# MERGE

- Syntax

- **MERGE** INTO <table> *-- new keyword **MERGE***
  - **USING** <table\_or\_join> *-- source of data for merging*  
ON <search\_condition> *-- how to find target row*
  - [WHEN **MATCHED** THEN *-- UPDATE query specification*  
UPDATE SET col1 = val1, ..., colN = valN]
  - [WHEN **NOT MATCHED** THEN *-- INSERT query specification*  
INSERT [(col1, ..., colN)] VALUES (val1, ..., valN)]
- Both **INSERT** and **UPDATE** query specifications are optional but at least one of them must be defined



# MERGE

- **MERGE** statement is handled by the engine like a more complex construction below :

```
FOR SELECT <table>.RDB$DBKEY
    FROM <table_or_join> LEFT JOIN <table>
        ON <search_condition>
    INTO :table_dbkey
DO IF (:table_dbkey IS NULL)
    THEN INSERT INTO <table> ...
    ELSE UPDATE <table> SET ...
        WHERE RDB$DBKEY = :table_dbkey;
```

- As **UPDATE OR INSERT** statement **MERGE** is regular SQL statement and can be used directly anywhere (not only inside procedures)



# MERGE

## MERGE example

```
CREATE TABLE AMOUNTS
(
  GOODID INT,
  SUMMA NUMERIC(18, 4),

  CONSTRAINT PK_AMOUNTS
    PRIMARY KEY (GOODID)
);
```

```
CREATE TABLE ENTRIES
(
  GOODID INT,
  DT DATE,
  DBT NUMERIC(18, 4),
  CRD NUMERIC(18, 4),

  CONSTRAINT PK_ENTRIES
    PRIMARY KEY (GOODID, DT)
);
```

```
MERGE INTO AMOUNTS USING ENTRIES E
  ON GOODID = E.GOODID
  WHEN MATCHED
    THEN UPDATE SET SUMMA = SUMMA + E.DBT - E.CRD
  WHEN NOT MATCHED
    THEN INSERT (GOODID, SUMMA)
    VALUES (E.GOODID, E.DBT - E.CRD)
```

ENTRIES			
GOODID	DT	DBT	CRD
1	01.01.2006	10.00	0.00
2	03.01.2006	0.00	4.00
2	02.01.2006	10.00	0.00
2	01.01.2006	15.00	0.00
1	05.01.2006	0.00	3.00

AMOUNTS	
GOODID	SUMMA
1	7.00
2	21.00



# NEW BUILT-IN FUNCTIONS

---

- **Mathematical**

- ABS, MOD, SIGN, CEIL, CEILING, FLOOR, ROUND, TRUNC, PI
- LOG, LOG10, LN, POWER, EXP, SQRT
- COS, COSH, ACOS, SIN, SINH, ASIN, TAN, TANH, ATAN, ATAN2, COT

- **Bit logic**

- BIN\_AND, BIN\_OR, BIN\_SHL, BIN\_SHR, BIN\_XOR

- **Datetime arithmetics**

- DATEADD, DATEDIFF

- **String manipulations**

- ASCII\_CHAR, ASCII\_VAL
- LEFT, RIGHT, LPAD, RPAD, POSITION, REPLACE, OVERLAY, REVERSE

- **Other**

- DECODE, MAXVALUE, MINVALUE
- GEN\_UUID, HASH, RAND



# Questions ?

---

