

# MySQL New Features Workshop v2.0

OSCON 2004

Zak Greant  
MySQL AB Community Advocate

## Tutorial Overview

- Feel free to ask questions during the tutorial
- 
- Slides will be available:
  - <http://dev.mysql.com/tech-resources/presentations/>
- The slides and the MySQL 5.0.1 sources are:
  - Shared via wireless + Rendezvous
  - Available on this USB key drive
  - Available on this CD

# MySQL Development Cycle

- **Maintained Releases**
  - 3.23.x Deprecated
- **Production Release**
  - 4.0.x Stable
- **Development Releases**
  - 4.1.x Beta (use for new development)
  - 5.0.x Alpha (feature previews and testing)
  - 5.1.x Leading Edge (feature previews and testing)

## Installing the Good Stuff

- Convenient installs
  - <http://dev.mysql.com/downloads/>
  - Source, Binaries, RPMs and installers
- Quick access to fresh code
  - <http://snaps.mysql.com>
  - Not perfectly up-to-date, but still very fresh
  - Need to compile
  - More instructions at <http://mysql.com/install> source tree
- Fresh from the developers
  - Get the source from our Bitkeeper tree
  - Instructions at <http://mysql.com/install> source tree

## Major Features in 4.1

- Enhanced binary protocol
- Client-integrated help system
- Better I18N support (including Unicode support)
- 
- Prepared statements
- Spatial data support
- Subqueries
- Warning system

## Major Features in 5.0

- Cursors
- Stored Procedures
- (Updatable) Views

## MySQL Cluster

- Main memory highly-available clustered database
  - Complemented with logging to disk
  - Database distributed over many nodes
  - Synchronous replication between nodes
  - Asynchronous replication between clusters
  - Database automatically partitioned over the nodes
  - Fail-over capability in case of node failure
- Update in one MySQL server, immediately see result in another
- Support for high update loads (as well as very high read loads)
- Former property of Ericsson
- Released under the GPL and sold under a proprietary license

## Questions for the Audience

- What databases do you use?
- What version of MySQL do you use?
- What languages/interfaces do you use with MySQL?
- Have you tried MySQL Cluster?
- How many of you know that MySQL is a company?
- How many of you are interested in licensing issues?

# MySQL 4.1

(MySQL 4.1.2+)

# Major New Features in MySQL 4.1

**Help, Enhanced Internationalization, Prepared  
Statements, Subqueries, Warnings**

## An Integrated Help System

- Fetch terse help on MySQL functionality from the client
- MySQL clients 4.1+ can query the `mysql.help_%` tables on any MySQL server
- The help tables are populated with the `fill_help_tables` script, which parses the relevant data from `manual.texti`
- Use `HELP ...` to access the help for a given topic
- See a list of topics with `HELP CONTENTS`

## Viewing the Help Topics

```
mysql> HELP CONTENTS
You asked help about help category: "Contents"
For a more information type 'help <item>' where
item is one of the following
categories :
  Administration
  Column Types
  Data Definition
  Data Manipulation
  Functions
  Geographic features
  Transactions
```

## An Integrated Help System (cont.)

- Some topic names are quite long.
- You can use the `_` and `%` wildcards to match strings and type less
- ie. `HELP _ELECT` or `HELP DATA MAN%`
- No help for 5.0 in the Bitkeeper trees
  - If strongly desired, grab `fill_help_tables` from a packaged release.

## Viewing a Particular Help Topic

```
mysql> HELP DATA MAN%
You asked help about help category: "Data
Manipulation"
For a more information type 'help <item>' where
item is one of the following
topics :
    CACHE INDEX
    CHANGE MASTER TO
    DELETE
    DO
    EXPLAIN
    FLUSH
    HANDLER
    INSERT DELAYED
    INSERT INTO
    INSERT SELECT
    JOIN
    . . .
```

## Help Content Still a Bit Flawed

- The current `fill_help_tables` script extracts data from the texinfo-based manual
- It leaves in some texinfo format sequences that should not be confused with SQL syntax
- Note the `@var{...}` bits in the following sample
- The future version of `fill_help_tables` will pull data from the new docbook-based manual

## Viewing a Specific Help Topic

```
mysql> HELP SEL%
Name: 'SELECT'
Description:
SELECT is used to retrieve rows selected from one
or more tables.
Support for UNION statements and subqueries is
available as of MySQL
4.0 and 4.1, respectively.
See [UNION, , UNION] and [Subqueries].

--- Each @var{select_expr} indicates a column you
want to retrieve.

--- @var{table_references} indicates the table or
tables from which to retrieve rows.
Its syntax is described in [JOIN, , JOIN].

...
```

## Robust Internationalization Support

- Store and manipulate strings with different character sets and/or collations in the same server, the same database, the same table or even the same query
- A character set is a set of symbols and encodings
- A collation is a set of rules for sorting a character set
- 
- Supported in InnoDB, MEMORY and MyISAM storage engines
- 
- Includes Unicode support
  - UCS-2 (but cannot yet be used as a client character set)
  - UTF8 (but no 4 byte sequences yet)

## A Character Set For A Four-letter Alphabet

- Alphabet:            'A',    'B',    'a',    'b'
- Encodings:            0        1        2        3
- The Character Set:
  - A 0
  - B 1
  - a 2
  - b 3

## A BINARY Collation for our Sample

- ... WHERE 'A' < 'B'
  - the comparison returns true
  - the encoding of 'A' (0) is less than the encoding of 'B' (1)
- ... WHERE 'A' = 'a'
  - the comparison returns false
  - the encoding of 'A' (0) is different than the encoding of 'a' (2)
- Non-binary collations included transformative rules to alter the comparison
  - “ü” = “ue”
  - “A” = “a”
  - “A” = “eh” // latin1\_canadian

## Character String Attributes

- Character strings have:
  - a character set
  - a length in characters
  - a length in bits
  - a coercibility attribute
  - collation (usually)
- Find the character set using the CHARSET function
- Find the character length using the CHAR\_LENGTH function
- Find the binary length using the BIT\_LENGTH function
- Find the collation using the COLLATION function

## Getting String Attributes

```
mysql> SET @chr_str = CONVERT('ABC' USING utf8);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT CHARSET(@chr_str) AS `character set`,
->        CHAR_LENGTH(@chr_str) AS characters,
->        BIT_LENGTH(@chr_str) AS bits,
->        COLLATION(@chr_str) AS collation;
```

character set	characters	bits	collation
utf8	3	24	utf8_general_ci

```
1 row in set (0.00 sec)
```

## Character String Attributes (cont.)

- The coercibility attribute contains information on:
  - how a character string got its collation
  - how strings with different collations can be compared
- Use the COERCIBILITY function to find the coercibility attribute for a character string
- The coercibility attribute may be one of:
  - COERCIBLE (3) - meaning that the string uses the default collation for its character set
  - EXPLICIT (0) - meaning that it has an explicit collation
  - IMPLICIT (2) - meaning that it uses the appropriate default collation for the column
  - NO COLLATION (1) - meaning that it has no collation - perhaps as the result of a concatenation operation between strings with different collations

## Finding a Strings Coercibility Attribute

```
mysql> SELECT COERCIBILITY('abc' COLLATE
-> latin1_swedish_ci);
```

COERCIBILITY('abc' COLLATE latin1_swedish_ci)
0

```
1 row in set (0.00 sec)
```

## Collations in MySQL

- One character set can have many collations
  - latin1 has latin1\_bin, latin1\_german1\_ci, latin1\_german2\_ci, etc.
- A character string will have exactly zero or one default collation
  - a particular latin1 string could use latin1\_swedish\_ci
- Collations can only be used for the corresponding character set

## Viewing Available Character Sets and Collations

- Use `SHOW CHARACTER SET` to show the available character sets on a MySQL server
- Use `SHOW COLLATION` to show the available collations on a MySQL server
  - Note that the collation names generally end in suffixes that indicate if they are case-sensitive (`_cs`), case-insensitive (`_ci`) or binary (`_bin`) collations

## Viewing the Available Character Sets

```
mysql> SHOW CHARACTER SET\G
***** 1. row *****
      Charset: dec8
      Description: DEC West European
      Default collation: dec8_swedish_ci
      Maxlen: 1
***** 2. row *****
      Charset: cp850
      Description: DOS West European
      Default collation: cp850_general_ci
      Maxlen: 1
***** 3. row *****
      Charset: hp8
      Description: HP West European
      Default collation: hp8_english_ci
      Maxlen: 1
***** 4. row *****
...

```

## Viewing the Available Collations

```
mysql> SHOW COLLATION LIKE 'latin%'\G
***** 1. row *****
Collation: latin1_german1_ci
  Charset: latin1
    Id: 5
  Default:
  Compiled:
  Sortlen: 0
***** 2. row *****
Collation: latin1_swedish_ci
  Charset: latin1
    Id: 8
  Default: Yes
  Compiled: Yes
  Sortlen: 1
***** 3. row *****
...
```

## Setting Default Character Sets and Collations

- Defaults character sets and collations can be set at many different levels from per-server down to per-column
- In general, more granular defaults take precedence
- The MySQL server generally has a generically useful default set.
- Individual database and tables then have appropriate defaults set for the data stored.
- Character sets and collations can be mixed within a table
- Character strings can also be cast into different character sets and collations

# Setting Default Character Set and Collations When Starting MySQL

```
shell> mysqld \  
--default-character-set=latin1 \  
--default-collation=latin1_swedish_ci
```

```
vi my.cnf  
...  
[mysqld]  
...  
default-character-set=latin1  
default-collation=latin1_swedish_ci  
...
```

## Per-table Character Set and Collation Defaults

```
mysql> CREATE TABLE trivial (  
->   name_hebrew  VARCHAR(64)  
->               CHARACTER SET hebrew,  
->   name_english VARCHAR(64)  
->               COLLATE latin1_general_cs,  
-> ) DEFAULT CHARACTER SET latin1;  
Query OK, 0 rows affected (0.16 sec)
```

# Coercing a Character String to use a Different Character Set and Collation

```
mysql> SELECT CONVERT(_latin1'Müller' USING utf8);
+-----+
| CONVERT(_latin1'Müller' USING utf8) |
+-----+
| Müller                               |
+-----+
1 row in set (0.00 sec)
```

```
mysql> INSERT INTO utf8table (utf8column)
-> SELECT CONVERT(latin1field USING utf8)
-> FROM latin1table;
```

## Connection Character Set

- Can be changed with statements:
  - SET NAMES latin1;
  - SET NAMES latin1 COLLATE latin1\_german2\_ci;
  - SET NAMES DEFAULT
  - SET COLLATION latin1\_german2\_ci;
- Character set chosen as follows:
  - Initially, it's the system character set.
  - When a client connects, it's the client's character set.
  - If SET NAMES 'X' is executed, then it's X.

## Character String Literals

- New syntax options:
  - `INSERT INTO t1 VALUES (_latin1 'a');`
  - `INSERT INTO t1 VALUES ('a' COLLATE latin1_german2_ci);`
- Character set chosen as follows:
  - If `_x` is specified, then it's `x`
  - Otherwise, it's the connection character set
  - Collation chosen as follows:
    - If `COLLATE x` is specified, then it's `x`
    - Otherwise, it's the default collation of the literal's character set

## More Examples of Character String Literals

```
SELECT _latin1'Müller' COLLATE latin1_german2_ci ;  
SELECT _latin1'Müller';  
SELECT 'Müller';  
SELECT * FROM Table1 WHERE column1 = 'Müller';
```

## Special case

- No problems with collations
  - `SELECT X FROM T1 ORDER BY X;`
  - `SELECT X FROM T1 WHERE X = X;`
  - `SELECT DISTINCT X FROM T1;`
- Special case!
  - `SELECT X FROM T1 WHERE X = 'Y';`
  - The collation from 'y' is taken from X

## ORDER BY with COLLATE

```
mysql> CREATE TABLE sample (name CHAR(10));  
Query OK, 0 rows affected (0.15 sec)  
  
mysql> INSERT sample VALUES ('MX Systems'),  
( 'Müller'), ('MySQL'), ('Muffler');  
Query OK, 4 rows affected (0.06 sec)  
Records: 4  Duplicates: 0  Warnings: 0
```

## ORDER BY with COLLATE (cont.)

```
mysql> SELECT name FROM collate_sample ORDER BY name  
COLLATE latin1_german1_ci;
```

```
+-----+  
| name |  
+-----+  
| Müller |  
| Muffler |  
| MX Systems |  
| MySQL |  
+-----+
```

```
4 rows in set (0.19 sec)
```

## ORDER BY with COLLATE (cont)

```
mysql> SELECT name FROM collate_sample ORDER BY
name COLLATE latin1_bin;
```

```
+-----+
| name  |
+-----+
| MX Systems |
| Muffler  |
| MySQL   |
| Müller  |
+-----+
```

```
4 rows in set (0.01 sec)
```

## Conversion with CONVERT

- The CONVERT function provides a way to convert data between different character sets.
- CONVERT(<expr> USING <transcoding name>)
  - SELECT CONVERT(\_latin1'Müller' USING utf8) ...
- SELECT CONVERT(latin1 field USING utf8) FROM latin1table;
- Another way to convert is with a CAST operator.
- CAST('a' AS CHAR(1) CHARACTER SET latin1)

# UTF8

- One Byte Needed For:
  - Basic Latin letters, digits and punctuation
- Two Bytes Needed For:
  - Extended Latin letters (with tilde, macron, acute, grave and other accents)
  - Cyrillic
  - Greek, Armenian, Hebrew, Arabic, etc.
- Three Bytes Needed For:
  - Korean, Chinese and Japanese ideographs

## Compatibility with the SQL Standard

Feature	Standard SQL?
separate character set / collation objects	YES
multiple character sets in one table	YES
COLLATE clause	YES
_introducer	YES
coercibility	YES

To compare other DBMS implementations of collations, see <http://www.dbazine.com/gulutzan1.html>

## More Information on Character Sets and Collations

- The MySQL Manual
  - <http://dev.mysql.com/doc/mysql/en/Charset.html>
- SQL-99 Complete, Really!
  - Peter Gultzan and Trudy Pelzer
  - ISBN 0-87930-568-1
- The SQL:2003 Standards
  - Purchase the standard from ISO at <http://www.iso.org>
  - Download a late draft from the Whitemarsh Information Systems Corporations at [http://www.wiscorp.com/sql/sql\\_2003\\_standard.zip](http://www.wiscorp.com/sql/sql_2003_standard.zip)

## Improved Client/Server Protocol

- New client/server communication protocol that features:
  - in-line zlib compression
  - lower overhead than the old protocol
  - when using prepared statements, all data is sent in the appropriate form - older versions of the protocol sent numeric data as characters
  - removes overhead of parsing/converting numeric data in both client and server compared to existing MySQL protocol
  - blob/clob data sent in chunks to server without storing them locally in the client end
- For details of implementations, see `internals.texti` in the `mysqldoc` module bitkeeper tree
  - see <http://mysql.com/install> source tree for instructions on using the bitkeeper tree

## Prepared statements

- Allows separation of query preparation from query execution
- Prepare Phase - the statement is first parsed, and later on this is executed one or more times using the statement “handle” that is returned from this phase.
- Binding Phase - if there are any parameters, then they are “bound” to local variables prior to execution.
- Execution Phase - statement executed efficiently, more efficiency realized over multiple executions
- Safer and Faster!

## Prepared Statements in PHP

```
# Connect to the database server
$link = mysqli_connect($host, $user, $pass, $db);

# Ask the server to prepare a query
$stmt = $link->prepare("SELECT name, population
FROM City");

# Bind the results to local variables
$stmt->bind_result($name, $population);

$stmt->execute();      # Execute the query

# Iterate over the query results
while( ! $t = $stmt->fetch() )
    printf("%-32s\t%s\n", $name, $population);

# Close the statement handle and connection
$stmt->close();
$link->close();
```

## More Prepared Statements in PHP

```
# Connect and create cart table
$stmt = $link->prepare("INSERT cart (name, price)
VALUES (?, ?)");
$stmt->bind_param(
    array(MYSQLI_BIND_STRING, MYSQLI_BIND_STRING),
    $name, $price);

$name = 'test';
$price = '2.00';

$stmt->execute();
$stmt->close();

$result = $link->query("SELECT name, price FROM
cart");
var_dump($result->fetch_row());
```

## Spatial Data

- Provides a powerful mechanism for manipulating point-based data (lines, polygons, ...)
- Good for maps, charts, vector data, ...
- Based on OpenGIS
- Needs more development and more supporting tools before it will be widely accessible

## Creating Spatial Columns

```
-- CREATE TABLE
mysql> CREATE TABLE geom (p1 GEOMETRY);
Query OK, 0 rows affected (0.02 sec)

-- ALTER TABLE
mysql> ALTER TABLE geom ADD p2 POINT;
Query OK, 0 rows affected (0.00 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

## Populating Spatial Columns

```
-- Using WKT functions:
INSERT geom VALUES(GeomFromText('POINT(1 1)'))

INSERT INTO geom VALUES
(GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(
5 5,7 5,7 7,5 7, 5 5))'))

INSERT INTO geom VALUES
(GeomFromText('GEOMETRYCOLLECTION(POINT(1 1),LINES
TRING(0 0,1 1,2 2,3 3,4 4))'))
INSERT INTO geom VALUES
(PointFromText('POINT(1 1)'))
INSERT INTO geom VALUES
(LineStringFromText('LINESTRING(0 0,1 1,2 2)'))
```

## Populating Spatial Columns (cont.)

```
-- Using WKB functions
mysql> INSERT INTO geom VALUES
(GeomFromWKB(0x01010000000000000000000000F03F00000000
000F03F));

-- Not for hand-coding by your average human!
```

## Spatial Indexes

- The most typical database searches:
  - A range of values above, below or between a set of criteria
  - An exact matches for a value
  - A partial match of a string
- The most typical spatial searches:
  - A point query. Searching for all objects that contain a given point.
  - A region query. Searching for all objects that overlap a given region.

## Using Spatial Indexes

```
-- Creating an index:
mysql> ALTER TABLE g ADD SPATIAL KEY(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376 Duplicates: 0 Warnings: 0

-- Using the index
mysql> SELECT fid,AsText(g) FROM g WHERE
-> MBRContains( GeomFromText(
-> 'Polygon((30000 15000,31000 15000,
->           31000 16000,30000 16000,
->           30000 15000))'), g);
.....
20 rows in set (0.00 sec) [Without index (0.46
sec)]
```

## Spatial Functions

- A wide variety of supporting functions
- Analyzing and reporting on the properties of geometric objects
- Converting between WKB and WKT formats
- See the MySQL manual for more information

## Subqueries

- A query embedded in another query
- The results of the subquery to be used as a scalar value, a list of values or a table.
- Provide a more natural way to combine data from tables compared to many styles of JOIN queries
- They provide queries that are structured

## Subquery Example

```
mysql> SELECT name, countrycode FROM city
-> WHERE city.population >
-> (SELECT MAX(population) FROM city
-> WHERE countrycode = 'CAN');
```

name	countrycode
Alger	DZA
Luanda	AGO
Buenos Aires	ARG
La Matanza	ARG
Yerevan	ARM
Sydney	AUS
:	:
Harare	ZWE

```
228 rows in set (0.02 sec)
```

## JOINS Instead of Subqueries?

- Many subqueries can be rewritten as JOINS
  - In fact, we used to recommend this before we implemented subqueries :)
- People often feel that using subqueries is more natural
- Take the following example: Given a city table and a country table, find all countries without cities
  - The above queries give exactly the same result

## Joins vs. Subqueries

```
mysql> SELECT name, code FROM country
-> WHERE NOT EXISTS
-> (SELECT * FROM city
-> WHERE countrycode = country.code);
```

```
mysql> SELECT country.name, country.code
-> FROM Country LEFT JOIN City on
-> (country.code=city.countrycode)
-> WHERE city.name is NULL;
```

-- Subquery is now faster (used to be much slower)

## Subqueries vs. JOINS

- JOINS can only be used in SELECT statements
  - Limited usage with UPDATE and DELETE
- Subqueries are supported in the following statements:
  - SELECT
  - UPDATE
  - INSERT
  - DELETE
  - REPLACE
  - DO
  - SET

## Derived Tables / Anonymous View

```
mysql> SELECT * FROM t1,  
-> (SELECT * FROM t2 WHERE a>5) t3  
-> WHERE t1.a = t3.a
```

## Expression Subqueries: Subqueries That Return a Value

```
-- Single value
mysql> SELECT t1.a,
      -> (SELECT a FROM t2 WHERE t2.b=t1.b) FROM t1

-- Row value
mysql> SELECT t1.a FROM t1 WHERE (a,b) =
      -> (SELECT c,d FROM t2 WHERE t2.e=t1.e)

-- Boolean EXISTS
mysql> SELECT t1.a FROM t1 WHERE EXISTS
      -> (SELECT * FROM t2 WHERE t2.a>t1.b)
```

## Expression Subqueries: Subqueries That Return a Value (cont.)

```
-- Boolean IN (scalar)
```

```
mysql> SELECT t1.a FROM t1 WHERE a IN (SELECT a  
FROM t2)
```

```
-- Boolean IN (ROW)
```

```
mysql> SELECT t1.a FROM t1 WHERE (a, b) IN  
(SELECT a, b FROM t2)
```

```
-- Boolean ALL/ANY/SOME (ROW)
```

```
mysql> SELECT t1.a FROM t1 WHERE a < ALL (SELECT a  
FROM t2))
```

## Scalar/single Row Subqueries

- `SELECT ... WHERE (a,b) = (SELECT c,d FROM ...)`
- In the above statement c and d are stored in the `Item_singlerow_subselect` cache
- If the subquery is correlated then the new values will be recomputed for each row of the outer query

## Error and Warnings System

- A past deficiency in MySQL was an inability to view all warnings and errors generated by a query
- In 4.1.1, the situation is much improved, with warnings generated for statements such as LOAD DATA INFILE and DML statements such as INSERT, UPDATE, CREATE TABLE, and ALTER TABLE
- Use SHOW WARNINGS/ERRORS to view warning and error messages
- Each query resets the warning/error message cache

## Error and Warnings System

```
mysql> SHOW ERRORS LIMIT 10;
```

```
mysql> SHOW COUNT(*) ERRORS;
```

```
mysql> SELECT @@warning_count;
```

```
mysql> SELECT @@max_error_count;
```

```
mysql> DROP TABLE IF EXISTS no_such_table;
```

```
mysql> SHOW WARNINGS;
```

Level	Code	Message
Note	1051	Unknown table 'no_such_table'

# Enhancements in MySQL 4.1

Enhanced CREATE TABLE, INSERT and UPDATE statements. B-tree indexes for MEMORY tables. New GROUP\_CONCAT function.

## CREATE TABLE Enhancements: Per Column Comments

- Tables can now have long comments on a per-column basis.
- Very handy for documentation that stays with a table
- Sometimes used to store meta-data for applications
  - Avoid this, as altering your comment requires altering the table. Frequent ALTER TABLE calls are still undesirable!
- Simply add COMMENT “comment text here” to the column definition.

## CREATE TABLE with Per-column Comments

```
mysql> CREATE TABLE trivial (  
->   id    SMALLINT UNSIGNED NOT NULL,  
->   name  VARCHAR(32) NOT NULL,  
->   fbid  TINYINT NOT NULL COMMENT "Foo Bar  
-> id for Bazco",  
->   ...  
-> );
```

## CREATE TABLE new\_table LIKE old\_table;

- Small extension, but very handy
- Creates a new table that has all of the same meta-data as the original table
  - indexes
  - comments
  - etc.
- Doesn't copy the data
- Syntax is: CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table2 LIKE table1.
- Currently fails for temporary tables

## Copying a Table Structure and Meta-Data

```
mysql> CREATE TEMPORARY TABLE privs LIKE  
mysql.user;
```

## GROUP\_CONCAT

- GROUP\_CONCAT() allows one or more groups or columns of data to be returned as a string, with an optional separator
- GROUP\_CONCAT([DISTINCT] expression, ...  
[ORDER BY column [ASC | DESC] [, ...]]  
[SEPARATOR str\_val])
- Default order is ascending (low to high)
- Default separator is a comma “,”

## Groups and GROUP\_CONCAT

```
mysql> SELECT countrycode,
->         GROUP_CONCAT(name SEPARATOR ", ")
-> FROM city
-> GROUP BY countrycode
-> ORDER BY countrycode
-> LIMIT 5;
```

countrycode	GROUP_CONCAT(name SEPARATOR ", ")
ABW	Oranjestad
AFG	Kabul, Qandahar, Herat, Mazar-e-Sharif
AGO	Luanda, Huambo, Lobito, Benguela, Namibe
AIA	South Hill, The Valley
ALB	Tirana

```
5 rows in set (0.04 sec)
```

## Basic Use of GROUP\_CONCAT

```
mysql> SELECT
-> GROUP_CONCAT(name SEPARATOR ", ") as names
-> FROM country\G
```

```
***** 1. row *****
names: Afghanistan, Netherlands, Netherlands
Antilles, Albania, Algeria, American Samoa,
Andorra, Angola, Anguilla, Antigua and Barbuda,
United Arab Emirates, Argentina, Armenia, Aruba,
Australia, Azerbaijan, Bahamas, Bahrain,
Bangladesh, Barbados, Belgium, Belize, Benin,
Bermuda, Bhutan, Bolivia, Bosnia and Herzegovina,
Botswana, Brazil, United Kingdom, Virgin Islands,
British, Brunei, Bulgaria, Burkina Faso, Burundi,
Cayman Islands, Chile, Cook Islands, Costa Rica
...
1 row in set, 1 warning (0.00 sec)
```

## GROUP\_CONCAT(): Increasing the String Length

```
mysql> SHOW VARIABLES LIKE 'group_concat_%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| group_concat_max_len | 1024 |
+-----+-----+
1 row in set (0.05 sec)
```

```
mysql> SET @@group_concat_max_len = 10240;
Query OK, 0 rows affected (0.06 sec)
```

## GROUP\_CONCAT: Cleaning Up The Output

```
mysql> SELECT
  -> GROUP_CONCAT(DISTINCT name ORDER BY name
SEPARATOR ", ") as names
  -> FROM country\G

***** 1. row *****
names: Afghanistan, Albania, Algeria, American
Samoa, Andorra, Angola, Anguilla, Antarctica,
Antigua and Barbuda, Argentina, Armenia, Aruba,
Australia, Austria, Azerbaijan, Bahamas, Bahrain,
Bangladesh, Barbados, Belarus, Belgium, Belize,
Benin, Bermuda, Bhutan, Bolivia, Bosnia and
Herzegovina, Botswana, Bouvet Island, Brazil,
British Indian Ocean Territory, Brunei, Bulgaria,
Burkina Faso, Burundi, Cambodia, Cameroon, Canada
...
1 row in set (0.01 sec)
```

## GROUP\_CONCAT and Subqueries

```
mysql> SELECT CONCAT(name, ": ",
->   (SELECT
->     GROUP_CONCAT(DISTINCT name
->                   ORDER BY population
->                   DESC SEPARATOR ", ")
->   FROM city
->   WHERE city.countrycode = country.code)
-> ) AS county_and_cities
-> FROM country LIMIT 5\G
```

```
***** 1. row *****
county_and_cities: Afghanistan: Kabul, Qandahar,
Herat, Mazar-e-Sharif
```

```
...
```

```
5 rows in set (0.18 sec)
```

## B-tree Indexes for MEMORY tables

- Pre-4.1, the MEMORY storage engine could only use hash-type indexes
  - Fast for exact string matches
  - Not useful for range comparisons or sorting
- 4.1+, the MEMORY storage engine can also use standard B-tree indexes
  - Fast range and sort operations
- Add USING BTREE to your index definitions
- Can have both b-tree and hash type indexes on a column

## Using B-tree Indexes with Memory Tables

```
mysql> CREATE TABLE lookup(  
->   id INT,  
->   INDEX USING BTREE (id))  
->   ENGINE = MEMORY;
```

# MySQL 5.0

(MySQL 5.0.1+)

# Major New Features in MySQL 5.0

Cursors, Stored Procedures, Views

## Stored Procedures

- A collection of SQL statements stored on the server and callable by name
- Greater independence from the client application
- Better network performance / more server load
- More secure - keeps operations on data on the server
- Still a very alpha implementation

## Stored Procedure Example

```
CREATE PROCEDURE withdraw(  
    p_amount DECIMAL(6,2),  
    p_tellerid INTEGER,  
    p_custid INTEGER)  
MODIFIES SQL DATA  
BEGIN ATOMIC  
    UPDATE customers  
        SET balance=balance - p_amount;  
    UPDATE tellers  
        SET cashionhand=cashionhand - p_amount  
        WHERE tellerid = p_tellerid;  
    INSERT INTO transactions  
        VALUES (p_custid, p_tellerid, p_amount);  
END
```

## Rudimentary Cursors

- Simple cursors are supported inside stored procedures and functions.
- 
- Cursors allow easy server-side forwards and backwards iteration over a result set
- The syntax is as in embedded SQL.
- Cursors are currently asensitive, read-only, and non-scrolling (forward-only)
- Asensitive means that the server may or may not make a copy of its result table.

## Rudimentary Cursor Sample

```
CREATE PROCEDURE curdemo()  
BEGIN  
    DECLARE done INT DEFAULT 0;  
    DECLARE cur CURSOR FOR SELECT a, b FROM t;  
    DECLARE code, name CHAR(16);  
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'  
SET done = 1;  
    OPEN cur;  
  
    REPEAT  
        FETCH cur INTO code, name;  
        IF NOT done THEN  
            -- operate on code, name (or whatever...)  
        END IF;  
    UNTIL done END REPEAT;  
  
    CLOSE cur;  
END
```

## Views

- A new logical table created from a query
- Very useful for restricting users to a subset of data or for building a logical table from many other tables
- Like tables, it is visible to the permissions system
- Relies on the underlying table's indexes for efficiency
- Updatable!
- Managed much like table: CREATE VIEW, ALTER VIEW, DROP VIEW, etc.

## Creating a Simple View

```
mysql> CREATE VIEW canada AS SELECT id, name,
population, district FROM city WHERE countrycode =
'CAN';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT name, district FROM canada LIMIT 5;
```

name	district
Montréal	Québec
Calgary	Alberta
Toronto	Ontario
North York	Ontario
Winnipeg	Manitoba

```
5 rows in set (0.00 sec)
```

## Creating a View of a View

```
mysql> CREATE VIEW alberta AS SELECT name,  
population FROM canada WHERE district = 'Alberta';  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT * FROM alberta;
```

name	population
Calgary	768082
Edmonton	616306

```
2 rows in set (0.01 sec)
```

## Updating a View (and the Underlying Tables)

```
mysql> INSERT canada (name, district) VALUES
('Foo', 'Bar');
Query OK, 1 row affected (0.43 sec)
```

```
mysql> SELECT * FROM city WHERE name = 'Foo';
```

ID	Name	CountryCode	District	Population
4080	Foo		Bar	0

```
1 row in set (0.01 sec)
```

```
-- note the missing countrycode
```

## Creating an Alternate View of User Privileges

```
mysql> CREATE VIEW privs AS SELECT host, user,  
  (if(Select_priv      = 'Y', 1 << 0, 0) |  
  if(Insert_priv      = 'Y', 1 << 1, 0) |  
  if(Update_priv      = 'Y', 1 << 2, 0) |  
  if>Delete_priv      = 'Y', 1 << 3, 0) |  
  if(Create_priv      = 'Y', 1 << 4, 0) |  
  if(Drop_priv        = 'Y', 1 << 5, 0) |  
  if(Reload_priv      = 'Y', 1 << 6, 0) |  
  if(Shutdown_priv    = 'Y', 1 << 7, 0) |  
  if(Process_priv     = 'Y', 1 << 8, 0) |  
  
  ...  
  if>Show_view_priv   = 'Y', 1 << 22, 0)) AS privmap  
FROM mysql.user;
```

## Use the View

```
mysql> SELECT * FROM privs;
+-----+-----+-----+
| host          | user  | privmap |
+-----+-----+-----+
| localhost     | root  | 8388607 |
| towel.local   | root  | 8388607 |
| towel.local   |       | 0        |
| localhost     |       | 0        |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

## Showing the CREATE Statement for a View

```
mysql> SHOW CREATE VIEW canada\G
***** 1. row *****
      Table: canada
Create Table: CREATE VIEW test.canada AS select
`test`.`city`.`ID` AS `id`,`test`.`city`.`Name` AS
`name`,`test`.`city`.`Population` AS
`population`,`test`.`city`.`District` AS
`district` from `test`.`city` where
(`test`.`city`.`CountryCode` = _latin1'CAN')
1 row in set (0.00 sec)
```

# Enhancements in MySQL 5.0

Improvements to the Query Optimizer.  
Enhancements to SELECT statement.

## Greedy Query Optimization

- The Query Optimizer is a component of almost all DBMS system.
- It seeks the best query execution plan for a given query.
- Pre-5.0, the MySQL Query Optimizer does an exhaustive search of all possible combinations of tables within a join and selects the best plan
- The cost of finding the perfect plan can be very high - sometimes more than the query
- The new greedy optimizer instead looks for a plan that is good, but not necessarily perfect
- This approach produces much more consistent and predictable performance.

## SELECT ... INTO @var, ...

- Cleaner syntax to assign query results into user variables
- Variables can be local or global in scope
- System variables cannot be set

# Alternate Syntax to Set Variables From Within SELECT

```
mysql> SELECT countrycode, name
      -> INTO @code, @name
      -> FROM city LIMIT 1;
Query OK, 1 row affected (0.02 sec)
```

```
mysql> SELECT @code, @name;
```

```
+-----+-----+
| @code | @name |
+-----+-----+
| AFG   | Kabul |
+-----+-----+
1 row in set (0.00 sec)
```

# Changes in MySQL 5.0

**A few backwards compatibility breakers**

## Binary Log Replaces the Update Log (5.0)

- The update log has been removed from 5.0
- It is replaced by the binary log
- If `--log-update` is set, it will be translated to `--log-bin` and a warning message will be written to the error log.
- Setting `SQL_LOG_UPDATE` will silently set `SQL_LOG_BIN` instead (or do nothing if the server is explicitly started with `--log-bin`).

## Users Variables Now Case-Insensitive (5.0)

```
mysql> SET @Foo = 42;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @FOO;  
+-----+  
| @FOO |  
+-----+  
| 42   |  
+-----+  
1 row in set (0.00 sec)
```

```
mysql> SELECT @fOo;  
+-----+  
| @fOo |  
+-----+  
| 42   |  
+-----+  
1 row in set (0.00 sec)
```

# MySQL Cluster

**A Clustered Highly-available Main Memory Database**

## MySQL Cluster Overview

- Main memory
  - Complemented with logging to disk
- Clustered
  - Database distributed over many nodes
  - Synchronous replication between nodes
  - Database automatically partitioned over the nodes
  - Fail-over capability in case of node failure
- Update in one MySQL server, immediately see result in another
- Support for high update loads (as well as very high read loads)

## MySQL Cluster Value

- Bringing clustered data management to a broader range of customers
  - Affordable
  - Ease of use
  - Upgrade path from current MySQL installations
- Making high-availability main-stream
  - On-going standardization efforts assume off-the-shelf components
  - New database technology using main-memory storage with replication gives persistency and short fail-over times

## MySQL Cluster Features

- Transactions
- Synchronous Replication
- Auto-synch at restart of NDB node
- Recovery from checkpoints and logs at cluster crash
- On-line Backup
- Subscription to row changes
- Indexes (Unique hash and T-tree ordered)
- On-line Index build
- On-line SW Upgrade
- Integrated in MySQL 4.1

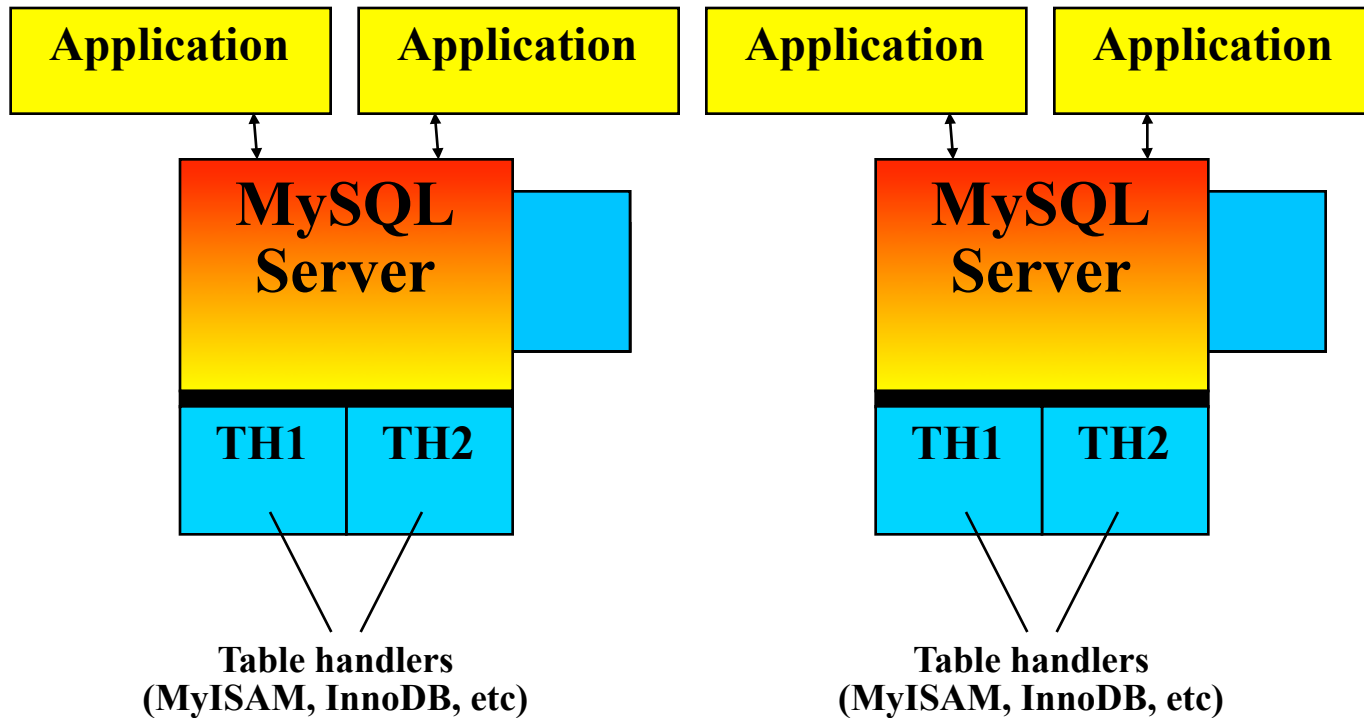
## Background

- Prototypes and research 1991-1996
- Draws from Ericsson's extensive experience in the Telecom/IP industry
- Development starts 1996 H2
- Version 0.1 August 1997 (primary keys and updates)
- Version 0.2 February 1998 (transaction support)
- Version 0.3 October 1998 (Cluster crash support)
- Version 1.0 April 2001 (Perl DBI, scan, ..)
- Version 1.1 August 2001 (production release)
- Version 1.41 May 2002 (Node Recovery)
- Version 2.11 May 2003 (Online Backup, Unique Index)
- Version 2.12.3 February 2004 (production release)
- Version 3.4.4 14th April 2004 (MySQL integration, Ordered Index, Online software Upgrade)

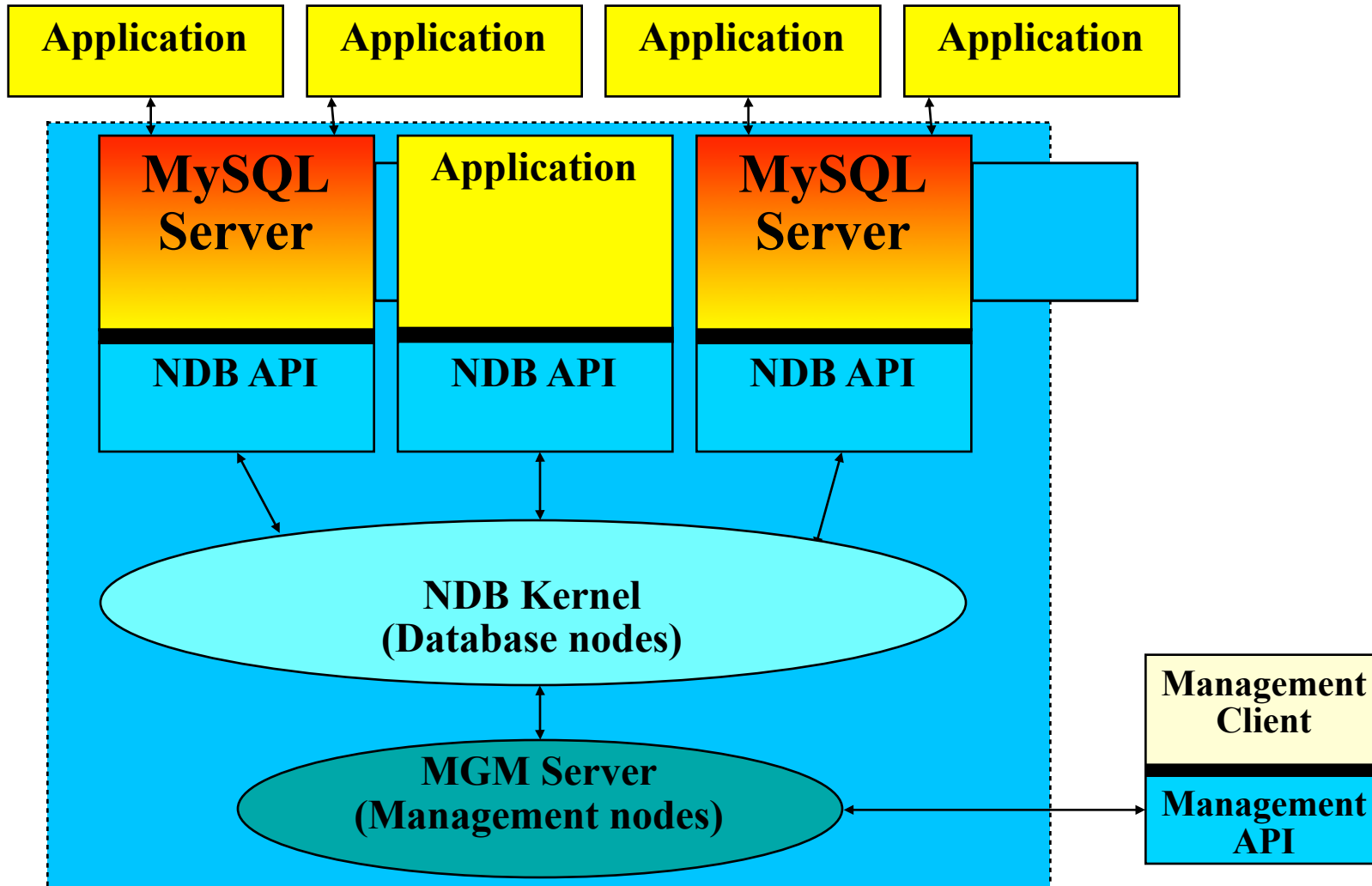
## Telecom Requirements for databases

- Availability
  - 99.999% (<5 min/year) or 99.9999%
- Performance
  - Response time (usually 5-10 ms)
- Throughput  $\geq 10.000$  transactions/sec
- Scalability
  - Many applications in distributed architecture
  - Cost-effective
- SQL for management apps, Fast interface for traffic apps, Unix/Windows/RT OS's

# MySQL "as you know it"



# MySQL Cluster Architecture



# Data Distribution

Horizontal fragmentation  
of Table 1 (4 fragments)

Fragments distributed  
on nodes

	Pnr	AccNo	Val	\$\$
<b>F1</b>				
<b>F2</b>				
<b>F3</b>				
<b>F4</b>				
<b>Table 1</b>				

2 copies of data

**F<sub>x</sub>** – primary replica  
**F<sub>x</sub>** – secondary replica

## Physical configuration

Dual CPU

Dual CPU

**Node 1**

**Node 2**

**Node 3**

**Node 4**

TCP/IP or SCI

## Logical configuration

Node group 1

**F1**  
**F3**

Node 1

**F1**  
**F3**

Node 2

Node group 2

**F2**  
**F4**

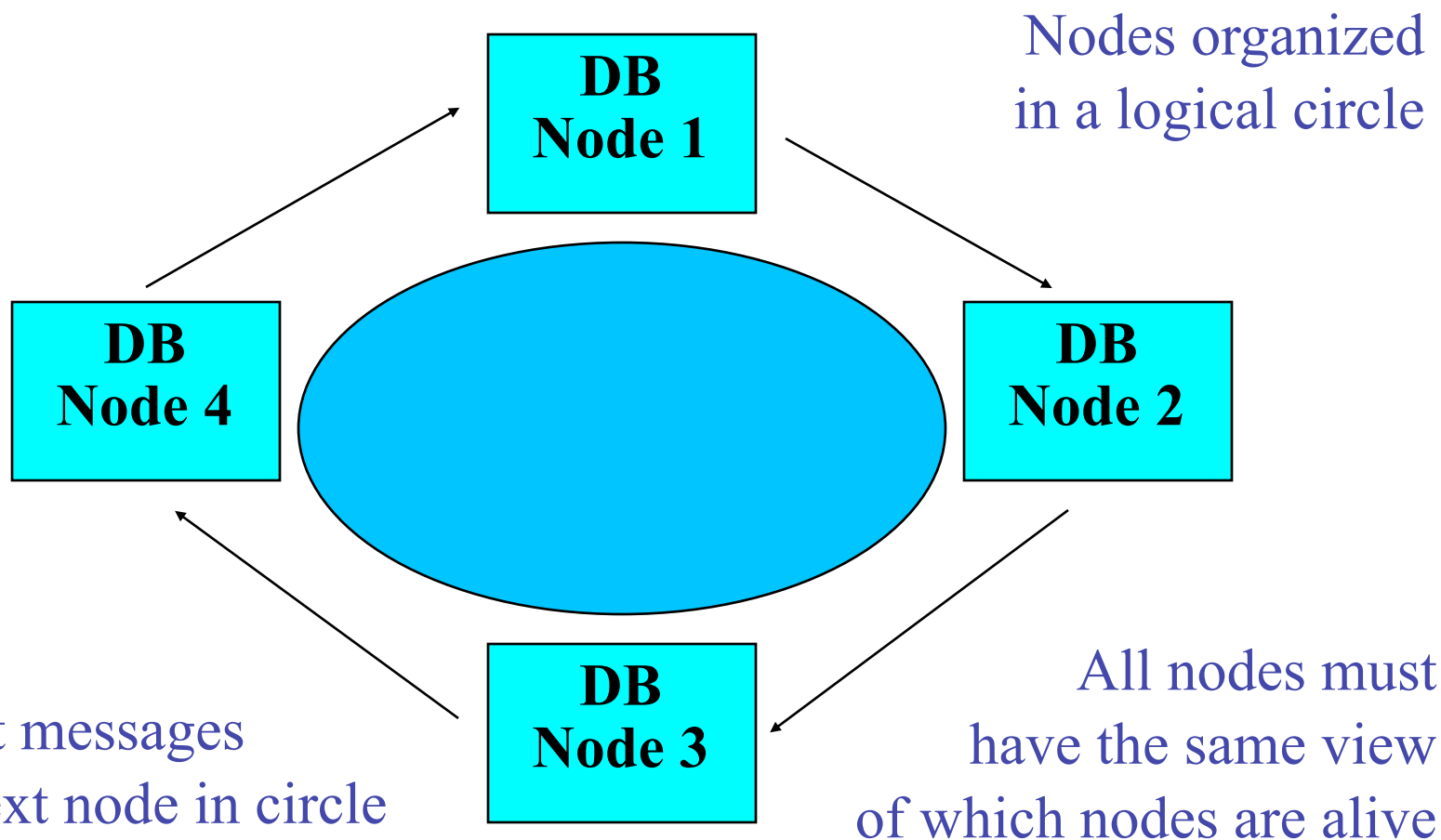
Node 3

**F2**  
**F4**

Node 4

**NDB Cluster : 4 node configuration on 2 dual processor machines**

# Failure detection: Heartbeats, lost connections



# Questions?

Thank You!