

MySQL and Progress Toward The SQL Standard

7. Juni 2004

Rechtlicher Hinweis

Dieser Beitrag ist lizenziert unter der Creative Commons License.

Zusammenfassung

In the last two years, MySQL AB has tried to adapt its product to conform with the ANSI/ISO SQL standards. We will see what it has done, and how far it has to go.

ANSI and ISO have published several versions of the SQL standard. The latest is SQL:2003. We'll start by looking at what's new in it.

MySQL version 4.1, and the new MySQL 5.0, have new "standard" features. The most notable are subqueries and stored procedures and views, but there are many small things that we have introduced without fanfare. For example: error codes with sqlstate, collate clauses, schema syntax, and better integrity checking.

I'll try to be frank about the things that we don't (yet) do, and our plans.

1. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
 2. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License.
 3. "Licensor" means the individual or entity that offers the Work under the terms of this License.
 4. "Original Author" means the individual or entity who created the Work.
 5. "Work" means the copyrightable work of authorship offered under the terms of this License.
 6. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
-
1. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
 2. to create and reproduce Derivative Works;
 3. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;
 4. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works;

1. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.
 2. You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.
 3. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author, or Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.
1. By offering the Work for public release under this License, Licensor represents and warrants that, to the best of Licensor's knowledge after reasonable inquiry:
 - (a) Licensor has secured all rights in the Work necessary to grant the license rights hereunder and to permit the lawful exercise of the rights granted hereunder without You having any obligation to pay any royalties, compulsory license fees, residuals or any other payments;
 - (b) The Work does not infringe the copyright, trademark, publicity rights, common law rights or any other right of any third party or constitute defamation, invasion of privacy or other tortious injury to any third party.
 2. EXCEPT AS EXPRESSLY STATED IN THIS LICENSE OR OTHERWISE AGREED IN WRITING OR REQUIRED BY APPLICABLE LAW, THE WORK IS LICENSED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES REGARDING THE CONTENTS OR ACCURACY OF THE WORK.

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.
1. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
2. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
3. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
4. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
5. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

1 Slide 1

1.1 MySQL and Progress Toward the SQL Standard

Presented By:
Peter Gulutzan
Software Architect
MySQL AB

1.1.1 Text:

Hello. I'm Peter Gulutzan, MySQL Software Architect. Today I'll be talking to you about the work MySQL has been doing to provide you with ever greater compatibility with the SQL Standard.

If you heard me talk at the MySQL User Conference last year, some of this will seem familiar. That's because I preannounced some features in 2003, and now I'm postannouncing them. Either way, it's progress, and worth reporting.

2 Slide 2

2.1 Interruption Policy

Any questions?
Interrupt or wait till the end.

2.1.1 Text:

I have to go over a lot of ground today, but I'll try to talk slowly and deliberately. I want to encourage anyone who wants to interrupt with a question or comment. I'll try to watch for raised hands. If I fail to notice you, just shout.

But I want to beg you to stay on topic during the slide show or you'll utterly discombobulate me. So, if you don't understand what's on a slide that I'm showing, or what I'm talking about, do ask. But if you have some further question, then please hold it till the end and we'll see if we can fit it into general discussion.

3 Slide 3

3.1 Does Peter Gulutzan know anything about this?

Books: SQL99 Complete, Really and three others
Articles: dbazine.com, orafaq.net, tdan.com, informit.com ...
Job since February 1 2003: Software Architect, MySQL AB

3.1.1 Text:

I have two reasons for claiming that I'm qualified to give this talk.

The first reason is that I have studied the SQL Standard for years. I am the coauthor of a thousandpage book titled "SQL99 Complete, Really" which describes the SQL/Foundation and SQL/CLI portions of the 1999 version of the Standard. I have also written three other books, as well as numerous articles about the current SQL Standard, which you can find on dbazine.com and in The Data Administration Newsletter, in the German iX magazine, et cetera.

The second reason that I'm qualified to give this talk is that I work for MySQL AB. That alone tells you something about MySQL's attitude, since they wouldn't have SQL Standard specialists on staff if they didn't care about standard SQL.

Equally importantly, I've had a chance to see how the developers react when we bring up Standardsrelated issues.

4 Slide 4

4.1 How I Pronounce Words

ANSI

Stands For: American National Standards Institute

Pronounced: antsy

ISO

Stands For: International Organization For Standards

Pronounced: Eye Ess Oh
SQL
Stands For: nothing in particular
Pronounced: Ess Cue Ell

4.1.1 Text:

I'll start by introducing three terms.

The first term is ANSI. In the United States, the common term for the Standard is "ANSI SQL" because the American National Standards Institute publishes it.

Outside the U.S., you'll sometimes hear the term "ISO SQL" instead. The terms are interchangeable because ANSI and ISO publish almost exactly the same Standards.

As for the term SQL, I will avoid saying that it stands for Structured Query Language since it doesn't really any more. And when it comes to the pronunciation, I have researched this issue. Microsoft and Oracle people prefer to say Sequel, but the official ANSI/ISO Standard word is SQL (ess que ell).

Therefore, I can already say this about MySQL's progress toward the SQL Standard: when it comes to pronouncing it right, we're ahead of Microsoft and Oracle.

5 Slide 5

5.1 The SQL Standards

SQL86, SQL89
SQL92
SQL99, also called SQL:1999
SQL:2003 or ISO/IEC 90752:2003

United States of America (ANSI) Information technology
Database languages SQL Part 2: Foundation (SQL/Foundation)

<http://www.jtclsc32.org>

5.1.1 Text:

I guess that means I should be able to tell you what the SQL Standard is. But in fact, I can't. There are several versions, and more than one of these versions has a claim to being called "the Standard".

I didn't bring the current version of the Standard along to hold up, because SQL:2003 has eleven different parts and its total size is over 1300 pages in the Foundation part alone. There are several other sections, but today I'll be concentrating on just one, the SQL/Foundation.

6 Slide 6

6.1 Parts of the Standard

1. Framework
2. Foundation
3. CLI i.e. Call Level Interface
4. PSM i.e. Persistent Stored Modules
5. Host Language Bindings
6. OLAP i.e. OnLine Analytical Processing
9. MED i.e. Management of External Data
10. OLB i.e. Object Language Bindings
11. Schemata
13. JRT i.e. Java Runtime Bindings
14. XML

6.1.1 Text:

Before I do that, though, I'll just give a brief mention of the other parts of the Standard.

- Part one, the framework, is just introductory material.
- Part two, the foundation, is what I've focused on throughout this talk.
- Part three, the call level interface, is somewhat like our ODBC implementation.
- Part four, the persistent stored modules, is what we did within stored procedures.
- Part five, host language bindings, deals mainly with embedded SQL, which we don't support.
- Part six talks about standard SQL's extra features for online analytical processing. We're ignoring it for now, although MySQL does have grouping support.
- Part nine, we ignore, although we do have plugins for storage engines.
- Part ten, we ignore.
- Part eleven, information schema, is something we plan to implement.
- Parts thirteen and fourteen, we ignore.

As I said, the key is the giant foundation document, part two. But we do keep an eye on the rest, and I hope we'll soon be announcing something important with respect to one of these other parts, soon.

7 Slide 7

7.1 The Three Main Versions (I)

SQL92
SQL99
SQL:2003 (current Standard)

7.1.1 Text:

At this time, there are actually three versions of the SQL Standard that matter.

8 Slide 8

8.1 The Three Main Versions (II)

SQL92
The older Standard (11 years out of date)
Some still call SQL92 "the" Standard
Minimum compliance is called Entry Level SQL92

8.1.1 Text:

SQL92 still counts because some people didn't like its successor, SQL99.
For example, some of you have heard of Joe Celko, who writes on SQL topics, and if so, you're aware that Mr. Celko prefers SQL92.

9 Slide 9

9.1 The Three Main Versions (III)

SQL99
The Standard until recently
Much, much bigger than SQL92
Minimum compliance is called Core SQL99

9.1.1 Text:

SQL99 is a much bigger affair, but most of the extra bulk is for the optional objectrelational features, which I'll only touch on lightly.
I should confess that the editor of this version of the SQL Standard Jim Melton says it should be called SQL:1999 rather than SQL99, but I believe that's unofficial.

10 Slide 10

10.1 The Three Main Versions (IV)

SQL:2003
The "current" Standard
Became official in December 2003
Minimum compliance is called Core SQL:2003

10.1.1 Text:

SQL:2003 became the official Standard just before Christmas of 2003, which was a relief to those of us who have been calling it SQL:2003 for a long time.

The main additions to the Standard in this newest version are in parts which deal with Java bindings and XML, but there are also a few new basic features. Surprisingly, some of those new features are already supported by the MySQL server, as we'll see.

Luckily, SQL:2003 is upward compatible with SQL99, which is upward compatible with SQL92. So usually I can just use the term quote the SQL Standard unquote to refer to all versions. However, when a feature applies to one version and not another, I'll be specific.

I only have enough time to discuss the minimum mandatory parts of these Standards. Each version of the Standard makes a clear distinction between what is mandatory and what's optional. For example, if your DBMS has all the mandatory features of SQL99, but no more, then the vendor may not say it has quote full compliance unquote but can say it has quote Core SQL99 compliance unquote.

11 Slide 11

11.1 Other Vendors' Claims

IBM DB2 claims Core SQL99 compliance

Oracle claims Core SQL99 compliance

MSSQL Server claims Entry Level SQL92 compliance

<http://dbazine.com/gulutzan3.html>

11.1.1 Text:

There are some tiny vendors who claim to be nearly at full compliance level, among them the company I used to work for, Ocelot. But among the major DBMS makers, SQL Standard compliance claims are modest. And in fact I would question even those modest claims.

So that I can avoid digressing into a discussion of IBM DB2, Oracle10g, and Microsoft SQL Server 2000, on this slide, I'm just showing a URL of an article that you can look at later. The article is titled SStandard SQLänd it will tell you how those three DBMSs rate if you look at their features and compare them with the SQL Standard requirements.

There's also a free white paper titledSSQL:2003 Standard Support in Oracle10g", but you'll have to get it from Oracle if you're interested.

12 Slide 12

12.1 MySQL's Goal

"One of our main goals with the product is to continue to work toward compliance with the SQL standard ..."

"We try to make MySQL Server follow the ANSI SQL standard ..."

The MySQL Reference Manual

12.1.1 Text:

Now, at MySQL, we're even more modest. We do say we quote follow unquote the Standard, but we mostly prefer to state it as a goal. To illustrate that, I've put up some statements that you'll find in the MySQL Reference Manual.

You can certainly see, from the number of times that we repeat it, that we care a lot about the SQL Standard and we're very vocal and public about stating that compliance with it is an objective of MySQL.

13 Slide 13

13.1 Tests For Compliance

The NIST Tests

NIST: National Institute of Standards and Technology

Tests apply only for SQL92

Feature List Tests

The Standard contains a list of mandatory features

13.1.1 Text:

Now that you know that MySQL AB's heart is in the right place, how can I demonstrate progress? I can think of two ways to check progress.

One way is to run MySQL against a publicly available set of tests that come from NIST, the U.S. National Institute of Standards and Technology. Unfortunately, the tests are old. They only look at SQL92 features. That, by the way, is the reason that some people don't trust SQL99. They say that without an independently made test set, claims of SQL99 compliance aren't provable.

Well, that's true, but at least a claim can be demonstrated. And I have to talk about modern standards, not the Standard of eleven or twelve years ago. So for the purposes of this talk I just took the feature lists that come with the SQL:2003 Standard documents.

The feature lists have the virtue that an independent body produces them. They have the defect that different people could decide differently whether a feature is supported or not. My own decision was that if a simple sample script runs as expected using the feature, then I'll count the feature as supported.

14 Slide 14

14.1 Testing MySQL's Compliance

Produced document listing Core SQL:2003 features

Checked off supported features

Noted areas that need work

14.1.1 Text:

To test compliance at MySQL, we conducted a review in early 2003.

We took the entire SQL Standard feature list for Core SQL:2003 and compared it with the features supported by MySQL versions 4.0, 4.1, and 5.0. For each feature, we made a sample script, a feature description, and a note of any deficiencies found. Then we dumped the result on the MySQL development staff.

Well, pretty quickly a reply came. As you probably all know, our developers love to read huge memoranda and scramble to adjust for every bit of feedback they get. As well as reviewing the document, two MySQL

staffers took the sample script and added it to the famous MySQL crash me tests, so that we were able to see how other DBMSs handle the SQL Standard feature list.

And that's my source of information.

15 Slide 15

15.1 The Standard Features List

Data Types
Identifiers
Operators
Subqueries & Joins
Constraints
Transaction Control
Set Functions
Schemas
INFORMATION_SCHEMA
Views
Grants & Revokes
...

15.1.1 Text:

Now, I could just repeat the list of all the features in Core SQL, in the order that the Standard document lists them. But I'm sure you're more interested in knowing the critical items, in a more organized form. So I'm going to go through the SQL features grouped by the main areas of interest.

The slide you see now is a table of contents for the next 29 slides.

16 Slide 16

16.1 Data Types: Numbers (I)

Exact (Fixed Point) Types
SMALLINT / INTEGER YES
BIGINT YES
DECIMAL / NUMERIC YES

Approximate (Floating Point) Types
REAL YES
FLOAT YES
DOUBLE PRECISION YES

16.1.1 Text:

Here is a list of the numeric data types that the SQL Standard requires a DBMS to support. Most of them have been around for a long time. Only BIGINT is new in SQL:2003.

The word YES means MySQL supports the data type. So no problem here.

17 Slide 17

17.1 Data Types: Numbers Scripts

```
CREATE TABLE Table1 (column1 NUMERIC(5));
INSERT INTO Table1 VALUES (999999);

CREATE TABLE Table2 (column1 REAL);
INSERT INTO Table2 VALUES (0.123456789E300);
... works fine if you use mysqld ansi
```

17.1.1 Text:

This slide has two test scripts which show slight departures from the SQL Standard.

In the first script, I've inserted a 6digit number into a NUMERIC(5) column.

In the second script, I've inserted a highprecision float into a REAL column, which should be a lowprecision data type. But actually the REAL data type is okay if the ansi switch is used.

18 Slide 18

18.1 Data Types: Temporal

```
Temporal (Date/Time) Types
DATE                YES
TIME                YES
TIMESTAMP          YES
INTERVAL            partial
```

18.1.1 Text:

Here is a list of the temporal data types that the SQL Standard requires. Again, there is a YES beside each data type that MySQL supports completely.

In our tests, we encountered some issues that prevent me from putting an unqualified YES beside the INTERVAL data type. However, INTERVAL support is an optional feature. It's not in the Core SQL requirements.

19 Slide 19

19.1 Data Types: Temporal DATE Scripts

```
CREATE TABLE Table1 (column1 DATE)
INSERT INTO Table1 VALUES (DATE '00010101')

Required date range is: 00010101 to 99991231
```

19.1.1 Text:

Now, on to the DATE data type.

The script that I'm showing here uses the SQL Standard's required syntax. MySQL is one of a minority of DBMSs that supports this precise syntax.

There is a strange detail here. The script shows an insertion of a date of January first, one AD. In fact we do not want to encourage arithmetic on such early dates, and so the MySQL Reference Manual says the minimum year is oneohohoh not ohohohone. So don't go home and try this.

But the fact is, the script worked. So MySQL passes.

20 Slide 20

20.1 Data Types: Temporal Scripts

```
CREATE TABLE Table1 (col1 DATE, col2 TIME);
INSERT INTO Table1 VALUES (CURRENT_DATE, CURRENT_TIME);

SELECT col1 + INTERVAL '15' DAY FROM Table1;
/* Succeeds */

SELECT col2 + INTERVAL '15' MINUTE FROM Table1;
/* Fails */
```

20.1.1 Text:

Now, some of you are accustomed to using MySQL's DATE_ADD and DATE_SUB functions for date arithmetic. Go right ahead, we've no intention of changing what exists already. In fact, we recommend that you use nonstandard functions when they're more convenient.

But I wanted to show the scripts on this slide because they are close to pure SQL Standard syntax, and MySQL does handle the datearithmic example correctly. But it does not handle the timearithmic example correctly at this time.

21 Slide 21

21.1 Data Types: Characters

Character String Types	
CHAR / VARCHAR	YES
NCHAR	NO
NCHAR VARYING	NO
CLOB /NCLOB	NO

21.1.1 Text:

Here we see the standard SQL character data types. MySQL supports CHAR and VARCHAR just fine. But MySQL only supports NCHAR and NCHAR VARYING by translating them to CHAR and VARCHAR with a fixed character set.

Support for the NCHAR, NCHAR VARYING, CLOB, and NCLOB data types is not a requirement for Core standard SQL. The fact that there is no national character string type and that you'd have to use TEXT instead of CLOB is thus not a standard SQL compliance problem.

22 Slide 22

22.1 Data Types: Characters Functions

CHAR_LENGTH	YES
OCTET_LENGTH	YES
POSITION	YES
SUBSTRING	YES
TRIM	YES
(concatenate)	YES (if ansi switch used)
UPPER / LOWER	YES
OVERLAY	YES

22.1.1 Text:

This slide shows the character string functions that the SQL Standard requires (except the ones related to transliteration). Again, MySQL supports them all. In fact, our support is better than most DBMSs'. Except for OVERLAY, the MySQL server supports the precise syntax that the Standard requires. Other DBMSs tend to support the function, but not the syntax.

23 Slide 23

23.1 Data Types: Characters Scripts

```
CREATE TABLE Table1 (column1 CHAR(2));
INSERT INTO Table1 VALUES ('X');
SELECT column1 || column1 FROM Table1;
```

Using ansi switch

Returns 'XX' instead of 'X X '

23.1.1 Text:

Here is a minor issue that we ran into when we checked out the character functions. MySQL sometimes trims trailing spaces when the SQL Standard doesn't require it. In this example case, though, we also found that there are other DBMSs which do exactly the same thing.

24 Slide 24

24.1 Data Types: Characters Character Sets & Collations

```
CREATE TABLE Table1 (column1 CHAR(5) CHARACTER SET latin1);
```

```
INSERT INTO Table1 VALUES (_latin1 'ABCDE');
SELECT column1 FROM Table1
ORDER BY column1 COLLATE latin1_german1_ci;
```

24.1.1 Text:

Beginning with version 4.1, MySQL has support for named character sets and collations, as required by standard SQL since SQL92.

The CREATE TABLE statement on this slide shows that MySQL lets you specify the character set for your data at the column level. The INSERT statement shows that you can specify the character set of a literal using what the SQL Standard calls an introducer. And the SELECT statement shows that you can specify a collating sequence for an operation that involves ordering.

Finally, notice that the character set and the collation are separate objects. These features are all done in the SQL Standard approved manner. In fact, this difficult set of features is well beyond the Core SQL requirement.

It's also well beyond the capability of other DBMSs. The last time I looked, which was about a year ago, DB2 and Sybase couldn't handle changes to character sets and collations for different columns, SQL Server had a nonstandard feature which sort of conflated the character set and the collation into one object, and Oracle's character set/collation support, while excellent, is also nonstandard.

In sum: among the major vendors, there is nobody who comes close to MySQL's support for SQL Standard-compliant character set and collation features.

25 Slide 25

25.1 Data Types: BIT

```
CREATE TABLE Table1 (column1 BIT(8));
```

25.1.1 Text:

I will pass quickly over the BIT data type. We are adding support for it, but meanwhile the ANSI and ISO Standards committee people have decided to drop this feature from the Standard.

Since it's not an SQL:2003 requirement, it's not a big deal. But we'll have it anyway.

26 Slide 26

26.1 Data Types: BOOLEAN

```
CREATE TABLE Table1 (column1 BOOL);
INSERT INTO Table1 VALUES (5 = 7);
/* i.e.: FALSE */
```

26.1.1 Text:

As for the BOOLEAN data type, we don't support it, but again this is not a Core SQL feature so that lack of support doesn't affect our claim of compliance.

Nevertheless, I do want to point out that MySQL can store and retrieve Boolean expressions, as the example shows. This is another thing that most DBMSs can't do, so it's a distinguishing feature for us.

27 Slide 27

27.1 Data Types: Other

```
Other Types
BLOB YES
UDT NO
```

Both of these data types are nonCore

27.1.1 Text:

To round out the list, I am noting here two final data types.

One is the BLOB data type, which of course we do support.

The other is the UDT, or userdefined type. MySQL doesn't support UDTs even in version 5.0. To support UDTs, we first need to support stored procedures. I'll come to that later.

28 Slide 28

28.1 Identifiers

```
CREATE TABLE "table1" ("column1" INT);
SELECT column1 FROM Table1;
-- MySQL return: 0 rows 0.00 seconds
-- Oracle return: ERROR: ORA00942: table or view does not exist
```

28.1.1 Text:

Here's an example of a statement that illustrates a lack of conformance to standard SQL.

For the pair of statements on the slide, which I ran in ansi mode, MySQL returns a result set. But when I ran the same script with Oracle, I got an error message saying that the table doesn't exist.

The bitter truth is that Oracle is right here and we are wrong. The SQL Standard says that the quoted identifier table1 (all lowercase) is not the same as the unquoted identifier Table1 (initial capital T). It's an issue having to do with case sensitivity.

I argued that we should change MySQL's behaviour. I lost. I think that this nicely illustrates the exception to our policy that we always will follow the Standard precisely. You can see why many people wouldn't want us to change this. I'll summarize at the end the points we are reluctant to change.

29 Slide 29

29.1 Operators

```
Arithmetic Operators
+ * / YES
```

```
Comparison Operators
= < <= > >= YES
```

LIKE YES

29.1.1 Text:

This slide shows the list of the Core standard SQL operators for arithmetic and comparison. MySQL handles them all. So we pass.
Next slide.

30 Slide 30

30.1 Subqueries

```
SELECT * FROM Table1 WHERE column1 =  
    (SELECT column1 FROM Table2);  
SELECT * FROM Table1 WHERE column1 > ANY  
    (SELECT column1 FROM Table2);  
SELECT * FROM Table1 WHERE column1 IN  
    (SELECT column1 FROM Table2);
```

30.1.1 Text:

Here you see some examples of what MySQL can now do with subqueries.

I've made sure to bring this up because subqueries are indeed a requirement for Core SQL Standard compliance. Their absence prior to this was a major reason that people criticized our SQL Standard commitment.

Well, that illustrates another aspect of MySQL's methods. We had a deficiency, we always admitted that it was something we should be criticized for, we determined to fix it, and we did fix it by adding this feature to version 4.1.

31 Slide 31

31.1 Joins

Traditional Join	YES
INNER JOIN	YES
CROSS JOIN	YES
LEFT JOIN	YES
RIGHT JOIN	YES
FULL JOIN	NO (not Core SQL)
ON clause	YES
USING clause	YES

31.1.1 Text:

Here is an incomplete list of the types of joins that the SQL Standard requires, and that MySQL supports. The requirement for support for FULL JOIN is, once again, not a requirement of Core SQL.

In this case, I have left out some complex examples which would show deficiencies in the current MySQL implementation. The development department tells me that there are known problems with outer joins within outer joins, and that other issues exist.

But we didn't actually run into those problems when we ran our tests. Or to put it another way: we passed, but we'll have to take the test again.

32 Slide 32

32.1 Constraints

	Core?	Supported?
PRIMARY KEY	Yes	YES
UNIQUE	Yes	YES
FOREIGN KEY	Yes	sometimes
NOT NULL	Yes	YES
CHECK	Yes	NO
TRIGGERS	No	NO

32.1.1 Text:

Now, we're getting into the group of features called "Constraints". I'm going to skip over the first two, because you probably know already that MySQL supports primary keys and unique keys. In fact MySQL supports them better than Microsoft does.

For example, we allow you to put more than one NULL value in a uniquekey column. Microsoft SQL Server doesn't allow that, so this is an example of a feature where we support the current SQL Standard and Microsoft doesn't.

Triggers is a nonCore feature, so let's look at the only Core SQL constraint type that MySQL doesn't support.

33 Slide 33

33.1 Unsupported Constraint Types

```
CREATE TABLE Table1 (column1 INT, CHECK (column1 > 5));
```

Missing a Core SQL requirement!

33.1.1 Text:

The only CoreSQL constraint type MySQL Server doesn't support is the CHECK clause.

This is the first time so far that I have had to put up a feature and say point blank: yes, we do not have that.

34 Slide 34

34.1 Constraints: Foreign Keys Flaws (I)

1. FOREIGN KEY (column_name) REFERENCES table_name
2. smallint_column REFERENCES pk_table (bigint_column)
3. unindexed_column REFERENCES pk_table (pk_column)

34.1.1 Text:

The MySQL implementation of foreign keys is okay. It actually does work, provided that you make sure to use the InnoDB storage engine rather than the default MyISAM storage engine. But there are some deficiencies.

One. You can't just say REFERENCES table name. You have to say REFERENCES table name followed by a column name list in parentheses. That's not a standard SQL restriction.

Two. You can't say that a smallint column REFERENCES a bigint column. With MySQL, the primary key and foreign key data types must be identical. That's not a standard SQL restriction either.

Three. You can't say that an unindexed column REFERENCES a primarykey column. You must create an index on the column before you can say that it's a foreign key. MySQL doesn't automatically create an index or decide that an index is unnecessary. This, too, is not a standard SQL restriction.

35 Slide 35

35.1 Constraints: Foreign Keys Flaws (II)

4. check the REFERENCES privilege
5. fk_column REFERENCES pk_table (non_unique_column)
6. DROP TABLE primary_key_table

35.1.1 Text:

Four. MySQL doesn't actually observe the REFERENCES privilege requirements, although it's possible to issue statements like GRANT REFERENCES ON table_name to peter, et cetera. That's not a standard way of checking privileges.

Five. The foreign key can reference a nonunique column. In other words, the so-called "primary key" table doesn't really have to have a primary key, or even a unique key. That's evading a standard SQL restriction, and it might seem like an extension, but it actually makes no sense.

Six. It's possible to drop the primarykey table while there's still a foreignkey table that's referencing it. That leaves the foreignkey table's contents pointing to nothing. The slang term for this is dangling. It's, of course, not the standard SQL way of doing things.

36 Slide 36

36.1 Constraints: Foreign Keys Fixes

```
The mistake with NO ACTION
-- People published on the web "there's a mistake"
-- We fixed the mistake
```

```
But we'll never catch up with the obsolete web critiques
-- So if you want to know uptodate criticism of MySQL
... JUST ASK US
```

36.1.1 Text:

On the other hand, there has been a fix in one area. There was a short period, maybe three or four months, during which the InnoDB programmers had an incorrect concept about what the NO ACTION clause is supposed to do.

The good news is, we were able to correct them and there's no problem now. So I guess that I could say that our quality assurance program is working, albeit a little slowly.

The bad news is, while the bad behaviour existed, various web site owners noticed it and published the information that, oh oh, MySQL is doing foreign keys wrong. Of course, those sites are totally out of date now, but once somebody publishes a criticism we can never catch up with and fix all of the sites after we fix the problem.

So, there is only one way to make sure that people have correct information about MySQL's flaws. That way is, we tell you ourselves what MySQL's flaws are. Consider what I just did on the previous slide. I gave you a flat statement about what is really wrong with foreign keys, today, currently, now. The reason that I do so is that MySQL is open source, of course. We figured out the benefits of having no secrets about our code.

But the big lesson is that if you want to find out what's wrong with MySQL, you just have to ask us. We know. You don't have to go looking for supposedly objective thirdparty information on some amateur's web site. Those guys just get their information by reading our manual or listening to talks like mine, and they are never up to date like we are. So believe me, the only reliable source for MySQL criticism is MySQL's site.

37 Slide 37

37.1 Transaction Control

```
START TRANSACTION
COMMIT
ROLLBACK
```

37.1.1 Text:

This slide shows more features that we added recently, and provides another illustration of the way MySQL AB works.

You all know that we pride ourselves on MySQL's speed when there is no transaction control. That stays. We're not going to slow your program down for the sake of COMMIT and ROLLBACK. But, for those who need it, we added transaction control, as options with the BDB and InnoDB table types.

The START TRANSACTION statement that I'm illustrating here is our new syntax for, well, for starting a transaction. The required syntax used to be BEGIN, but BEGIN isn't standard SQL and we thought it might cause parsing problems with our stored procedures. So I put in a request one day, saying: "Could we allow START TRANSACTION as alternate syntax here?"

I got a note back the following week, from the head of development saying: "Okay, that's done, we've implemented that."

And what that shows is: if you formulate your need the right way, and you ask nicely, and you explain that you're requesting a standard SQL syntax which will help others, you can get amazing turnaround time from the MySQL development staff.

38 Slide 38

38.1 Set Functions

```
CREATE TABLE Table1 (c1 INT);
INSERT INTO Table1 VALUES (NULL);
INSERT INTO Table1 VALUES (NULL);
SELECT AVG(c1), COUNT(*), MAX(c1), MIN(c1), SUM(c1) FROM Table1;
```

38.1.1 Text:

These are examples of standard SQL's set functions (also known as aggregate functions).

When testing set functions, we discovered one small flaw. When we run this SELECT statement, but with a one instead of a NULL in the first INSERT, the MySQL server returns the right answer: one. But MySQL doesn't also return an accompanying warning saying: "Warning: NULL values eliminated in set function".

This illustrates another general point. MySQL can do error returns very well, or MySQL can say "OKAY" very well. But MySQL doesn't do warnings very well.

As it happens, there are very few times that the SQL Standard requires warnings, so this isn't a frequently occurring problem for compliance.

39 Slide 39

39.1 Schemas

```
CREATE SCHEMA Schema1
  CREATE TABLE Table1 ...
  CREATE TABLE Table2 ...;

INSERT INTO Schema1.Table1 VALUES (1);
```

39.1.1 Text:

Now I'm going to start talking about the items that MySQL still needs to work on before we achieve Core SQL compliance.

The first item we had to decide is what we are going to do for schemas. For anyone who hasn't seen a schema statement before, a schema is just a container in which tables, privileges, and other SQL objects are stored. This example shows a CREATE SCHEMA statement which is making a schema named Schema1 and putting two tables inside it.

Now, some of you are looking at the INSERT statement and saying: "well duh, that's where we put the database name."

It's true that MySQL allows the syntax quote database name dot table name unquote. So we thought about this for a while, and decided that we really do support schemas, we just call them something else. That being the case, all we have to do now is substitute the word SCHEMA for the word DATABASE in our parser and we comply with standard SQL in this area. That hasn't been done yet, though.

40 Slide 40

40.1 INFORMATION_SCHEMA

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
instead of
SHOW TABLES
```

40.1.1 Text:

One of standard SQL's requirements is that there will always be a schema named INFORMATION_SCHEMA, and it will have metadata. That is, it will have information about the structure of the tables, the types of the columns, and so on.

You can, of course, already use MySQL's SHOW command to see metadata. However, we also plan to add a way to get the metadata information in table form, as the Standard requires.

41 Slide 41

41.1 Views

```
CREATE VIEW Views1 AS SELECT * FROM Table1;
```

```
:(
```

41.1.1 Text:

Here's another item we don't have. It's the only missing feature that we classify as major, but it will take some work. Views aren't in MySQL version 4.1 or even in MySQL version 5.0.

42 Slide 42

42.1 Grants & Revokes

```
GRANT SELECT, UPDATE ON Table1 TO peter;  
REVOKE UPDATE ON Table1 FROM peter;
```

42.1.1 Text:

The fact is, the SQL GRANT/REVOKE mechanisms are cumbersome. Our object has been to fulfill the intent, which is security. So we've concentrated on the underlay (encryption, ssh, identification of client address). That does mean that we have had to use some nonstandard syntax, otherwise we wouldn't have the support for security we feel our users need.

However, the standard SQL GRANT and REVOKE statements that you see on this slide actually work, so I'm including security on the list of supported features. If you want to noninclude security because it's usually necessary to use nonstandard identifiers, then count this one against us.

43 Slide 43

43.1 Error Returns: SQLSTATE

```
SQLSTATE  
SQL's basic piece of diagnostic information  
Core requirement
```

43.1.1 Text:

Here's another Core SQL requirement that MySQL has just begun to support in a standardcompliant fashion.

Our error return messages predate SQLSTATE. But this feature is one of the new and good things in MySQL version 4.1.

44 Slide 44

44.1 Error Returns: Warnings

```
CREATE TABLE Table1 (col1 CHAR(4), col2 SMALLINT, col3 DATE);
INSERT INTO Table1 (col1) VALUES ('abcde');
INSERT INTO Table1 (col2) VALUES (99999);
INSERT INTO Table1 (col3) VALUES (DATE '14477');
...
SHOW WARNINGS
```

44.1.1 Text:

The fact is, as I mentioned earlier, and as many people note in surprise or sorrow, MySQL doesn't type check strictly when you insert values into tables. But here's a new feature that first appeared about six months ago.

Now, if you insert a fivecharacter value in a fourcharacter column, you get a warning. Or, if you put a toobig number in a SMALLINT column, you get a warning. Or, if you put a ridiculous date in a DATE column, you get a warning. There are many examples of bad data input that MySQL does not detect, for example you can insert February 31 and the MySQL server won't complain yet. And of course we know that people don't just want to see warnings, they want to see error messages and they want to see that the statement gets rolled back.

Well, I don't say that's there. I just say that this is an example of progress along the way, that's happened within the last short while.

45 Slide 45

45.1 Summary Of Remaining Core SQL Deficiencies

Big Items

- CHECK Constraints

- Schemas

- Views

- INFORMATION_SCHEMA

Small Items

- etc.

45.1.1 Text:

So here are the reasons we can't say that MySQL version 4.1 meets the Core SQL compliance requirements. In the small items category, I've just said "et cetera" because I've now told you what they all are, and I think you'll agree that they truly are small.

For actual quote missing features unquote, I just count: Check clauses, Schemas (though we do kind of support something like them), Views, and INFORMATION_SCHEMA tables.

46 Slide 46

46.1 Things We Won't Do

Change Clike syntax additions
Add something that makes MySQL slower in default mode
Change case sensitivity for identifiers
Add errors for illegal assignments, usually

"The Great MySQL Caveat"

46.1.1 Text:

I'd now like to mention what I call the MySQL caveat. We are entering into a covenant with our users, promising that we will comply with all Core SQL requirements. But if you look closely at the written promise in the MySQL Reference Manual, you'll see that we add quote well, provided it doesn't wreck our users' lives unquote.

In practice, that's going to mean that we'll allow escape characters inside string literals, just like the C programming language does. And you know that we're very proud of the efficient way that MySQL runs. That's our crown jewel, and we won't jeopardize that for the sake of complying with a standard SQL feature.

I've already noted the third item on the slide, the subject of case sensitivity for quoted identifiers. And the final item is whether we should abort a statement and return an error message if, for example, you try to insert a 5character string into a 4character column. We won't do that even though it's a standard SQL requirement. One good reason is that, if we did, we could have halfdone transactions and thus ruin the integrity of a MyISAM table. So we can't really perform abortions as often as the SQL Standard would like.

47 Slide 47

47.1 Things We *DO* Do: Extensions to Core SQL (I)

IDENTITY	(SQL:2003 nonCore)
FLOOR, CEILING, etc.	(SQL:2003 nonCore)
CASE	(SQL:2003 nonCore)
CAST (partially)	(SQL:2003 nonCore)

47.1.1 Text:

Now, getting back to an upbeat note, I'll talk about some features supported by MySQL which are not Core SQL. That is, we have exceeded the minimum requirements in a few respects. This list doesn't include the items I've mentioned already, like named character sets.

One MySQL feature is an equivalent to the SQL:2003 IDENTITY columns, which we call AUTO_INCREMENT columns.

Another plus is that MySQL has a plethora of functions for numbers. We've always had them, because we decided from the beginning that SQL statements should have the same functionality that functions in the C

programming language have. Well, we were ahead of our time, because now many of these functions are in the SQL:2003 specification.

CASE and CAST are other nonCore features supported (at least partially) by MySQL.

48 Slide 48

48.1 Things We *DO* Do: Extensions to Core SQL (II)

F033	ALTER TABLE ... DROP COLUMN
F052	Intervals and datetime arithmetic
F111	Isolation levels other than serializable
F731	INSERT column privileges
T061	UCS support
T071	BIGINT
T141	SIMILAR

48.1.1 Text:

This slide, and the next one, show a fuller list of some standard nonCore features that MySQL is supporting now. This list is controversial, and I could put a question mark beside every item, but we do have support for ALTER TABLE DROP COLUMN, for some kinds of datetime arithmetic but not all, for READ UNCOMMITTED and READ COMMITTED isolation levels, for INSERT privileges that apply to only one column, for a UNICODE character set, for the BIGINT data type, and for something that works like the standard SQL SIMILAR operator namely MySQL's fulltext extension.

49 Slide 49

49.1 Things We *DO* Do: Extensions to Core SQL (III)

T171	LIKE clause in table definition
T172	AS subquery clause in table definition
T241	START TRANSACTION statement
T271	Savepoints
T281	SELECT privilege with column granularity
T351	Bracket SQL comments /* ... */
T441	ABS and MOD functions
T591	UNIQUE constraints of possibly null columns

49.1.1 Text:

MySQL also has support for statements like CREATE TABLE T2 LIKE T1 and CREATE TABLE T2 AS SELECT STAR FROM T1, for START TRANSACTION rather than BEGIN when a transaction starts, for rolling back to savepoints, for SELECT privileges that apply to only one column, for comments in the C style, for the absolute (ABS) and modulus (MOD) functions, and for UNIQUE constraints on columns that might contain nulls.

Quite a long list. But since all these features are nonCore, it doesn't help us to claim compliance with Core SQL.

Now, I could say lots more about MySQL's extensions. But I'd like to get on to the most exciting nonCore feature of them all: stored procedures.

50 Slide 50

50.1 Stored Procedures (I)

```
CREATE PROCEDURE Sp_procl
  @param1 INT
AS DECLARE @num1 INT
IF @param1 <> 0 SELECT @param1 = 1
UPDATE Table1 SET column1 = @param1

-- Source: SQL Performance Tuning
```

50.1.1 Text:

This slide shows what stored procedures do *not* look like. This syntax comes from an example for Microsoft SQL Server 2000, and I am just showing it so you'll see that other DBMSs don't follow the Standard at all when it comes to stored procedures.

51 Slide 51

51.1 Stored Procedures (II)

```
CREATE PROCEDURE Sp_procl (param1 INT)
MODIFIES SQL DATA
BEGIN
  DECLARE num1 INT;
  IF param1 <> 0 THEN SET param1 = 1;
  END IF;
  UPDATE Table1 SET column1 = param1;
END

-- Source: SQL Performance Tuning
```

51.1.1 Text:

This is what a stored procedure is *supposed* to look like. I've extracted these examples from a book on SQL that includes a lot of comparisons of different DBMSs.

The example on this slide was *not* the MySQL example. It was written without any knowledge of MySQL stored procedures. Instead, this is the book's example of pure ANSI/ISO syntax for stored procedures.

Well, if you try this with MySQL version 5.0 stored procedures, you'll see that you can run this example as is, except for the clause `MODIFIES SQL DATA`, which is a donothing clause anyway.

Since stored procedures are so recent to MySQL, I'm going to walk through this example because it's probably news to you.

52 Slide 52

52.1 Stored Procedures (III)

```
CREATE PROCEDURE Sp_procl (param1 INT)
MODIFIES SQL DATA
BEGIN
    DECLARE num1 INT;
    IF param1 <> 0 THEN SET param1 = 1;
    END IF;
    UPDATE Table1 SET column1 = param1;
END
```

52.1.1 Text:

The first words, "CREATE PROCEDURE Sp_procl", are analogous to CREATE TABLE. That is, I'm making a new object in the database, and it's a procedure rather than the other choice which was to make it a function. The procedure's name is Sp_procl.

53 Slide 53

53.1 Stored Procedures (IV)

```
CREATE PROCEDURE Sp_procl (param1 INT)
MODIFIES SQL DATA
BEGIN
    DECLARE num1 INT;
    IF param1 <> 0 THEN SET param1 = 1;
    END IF;
    UPDATE Table1 SET column1 = param1;
END
```

53.1.1 Text:

The next part, within parentheses, is the parameter list. In this case I'm saying that the procedure uses a parameter named param1. This describes what can be passed when the procedure is called. It happens to be an input parameter by default, but output parameters are also legal.

54 Slide 54

54.1 Stored Procedures (V)

```
CREATE PROCEDURE Sp_procl (param1 INT)
MODIFIES SQL DATA
BEGIN
    DECLARE num1 INT;
    IF param1 <> 0 THEN SET param1 = 1;
```

```
    END IF;
    UPDATE Table1 SET column1 = param1;
END
```

54.1.1 Text:

The clause "MODIFIES SQL DATA", which I've already noted is not supported by MySQL yet, is informative. It means that the procedure does a change to SQL data. Which is true, because the UPDATE statement inside the procedure changes SQL data.

55 Slide 55

55.1 Stored Procedures (VI)

```
CREATE PROCEDURE Sp_procl (param1 INT)
MODIFIES SQL DATA
BEGIN
    DECLARE num1 INT;
    IF param1 <> 0 THEN SET param1 = 1;
    END IF;
    UPDATE Table1 SET column1 = param1;
END
```

55.1.1 Text:

Looking next at the BEGIN, it matches up with the word END later on, which I've indented in the same place to indicate that BEGIN and END are paired up, as with Pascal. So there's what is called a BEGIN/END block here.

56 Slide 56

56.1 Stored Procedures (VII)

```
CREATE PROCEDURE Sp_procl (param1 INT)
MODIFIES SQL DATA
BEGIN
    DECLARE num1 INT;
    IF param1 <> 0 THEN SET param1 = 1;
    END IF;
    UPDATE Table1 SET column1 = param1;
END
```

56.1.1 Text:

You can declare variables within the BEGIN/END block. The DECLARE statement here is declaring a variable named num1 which is an integer. When you declare a variable, you can use all the size and data type options that you can use for a column, so there's no learning curve here. Simply put: what you can do for columns, you can do for variables too.

57 Slide 57

57.1 Stored Procedures (VIII)

```
CREATE PROCEDURE Sp_procl (param1 INT)
MODIFIES SQL DATA
BEGIN
    DECLARE num1 INT;
    IF param1 <> 0 THEN SET param1 = 1;
    END IF;
    UPDATE Table1 SET column1 = param1;
END
```

57.1.1 Text:

Next I've got an IFTHENEND IF statement block. This is by no means the only control statement that MySQL now supports. I just haven't illustrated all the ways that you can write a loop for a stored procedure.

58 Slide 58

58.1 Stored Procedures (IX)

```
CREATE PROCEDURE Sp_procl (param1 INT)
MODIFIES SQL DATA
BEGIN
    DECLARE num1 INT;
    IF param1 <> 0 THEN SET param1 = 1;
    END IF;
    UPDATE Table1 SET column1 = param1;
END
```

58.1.1 Text:

And finally, the procedure has an UPDATE statement which assigns a parameter value to a column in the database. You can use variable or parameter names pretty well anywhere where you could otherwise use a literal.

I have gone on at length about this procedure because it's useful to know, of course. But the important thing in all this is that every word I've said, every detail I've described, every bit of the description and punctuation, is standard, standard, standard SQL. This is the pure stuff.

59 Slide 59

59.1 We Make the UNION Strong (I)

	v4.0.0	v4.1.2
E07101 UNION DISTINCT table operator	mostly	YES
E07102 UNION ALL table operator	mostly	YES

E07103	EXCEPT DISTINCT table operator	NO	NO
E07105	Columns combined via table operators need not have exactly the same data type	YES	YES
E07106	Table operators in subqueries	NO	YES

59.1.1 Text:

At this point, I'd like to go into some detail about what has happened to the UNION operator in the period between the release of version 4.0.0 and version 4.1.2, a relatively short time. I usually find that this level of detail is unnecessary, but it is possible to do it unboringly once because you see what the details are for a particular feature, in this case feature E071 Basic query expressions, and you see that progress happens in pieces, not all at once.

By the way, a "table operator" is a standard SQL term for an operator that merges two SELECT statements. The most common table operator is UNION.

Originally, for feature E07101, UNION DISTINCT table operator, MySQL didn't support the word DISTINCT after the word UNION ... not a big deal since UNION is DISTINCT by default anyway, but we do support this feature now.

60 Slide 60

60.1 We Make the UNION Strong (II)

		v4.0.0	v4.1.2
E07101	UNION DISTINCT table operator	mostly	YES
E07102	UNION ALL table operator	mostly	YES
E07103	EXCEPT DISTINCT table operator	NO	NO
E07105	Columns combined via table operators need not have exactly the same data type	YES	YES
E07106	Table operators in subqueries	NO	YES

60.1.1 Text:

For feature E07102, UNION ALL table operator, MySQL once had a flaw due to not reading the Standard closely enough. If one SELECT returned values shorter than the other, there was truncation. Now there's expansion, instead.

61 Slide 61

61.1 We Make the UNION Strong (III)

		v4.0.0	v4.1.2
E07101	UNION DISTINCT table operator	mostly	YES
E07102	UNION ALL table operator	mostly	YES
E07103	EXCEPT DISTINCT table operator	NO	NO
E07105	Columns combined via table operators need not have exactly		

	the same data type	YES	YES
E07106	Table operators in subqueries	NO	YES

61.1.1 Text:

For feature E07103, EXCEPT DISTINCT table operator, MySQL fails, because we support only one table operator, UNION.

MySQL doesn't support EXCEPT (which is a Core SQL requirement) or INTERSECT (which is nonCore). So version 4.0.0 didn't have EXCEPT, and we don't still have it now: an example of why I keep using the word 'progress' rather than the word 'achievement'.

62 Slide 62

62.1 We Make the UNION Strong (IV)

		v4.0.0	v4.1.2
E07101	UNION DISTINCT table operator	mostly	YES
E07102	UNION ALL table operator	mostly	YES
E07103	EXCEPT DISTINCT table operator	NO	NO
E07105	Columns combined via table operators need not have exactly the same data type	YES	YES
E07106	Table operators in subqueries	NO	YES

62.1.1 Text:

For feature E07105, Columns combined via table operators need not have exactly the same data type, we started the period with that already in place. I do regard this as a slight advance, since we haven't got the same thing for joins.

63 Slide 63

63.1 We Make the UNION Strong (V)

		v4.0.0	v4.1.2
E07101	UNION DISTINCT table operator	mostly	YES
E07102	UNION ALL table operator	mostly	YES
E07103	EXCEPT DISTINCT table operator	NO	NO
E07105	Columns combined via table operators need not have exactly the same data type	YES	YES
E07106	Table operators in subqueries	NO	YES

63.1.1 Text:

Finally, for feature E07106, Table operators in subqueries, we naturally failed at the start because we didn't have subqueries period. Now we do, and MySQL does support both unions in subqueries, as well as subqueries in unions.

As a summary, I can say that MySQL has moved forward a little or a lot on three parts of feature E071, stayed the same on two parts, and is stuck only on one part, which is the table operator named EXCEPT. And for that final item, since other DBMSs don't handle the EXCEPT table operator either, I think that justifies me in saying that the deficiency is minor.

64 Slide 64

64.1 AVG

```
CREATE TABLE E091_01 (S1 FLOAT);
INSERT INTO E091_01 VALUES (1E2);
INSERT INTO E091_01 VALUES (1E2);
INSERT INTO E091_01 VALUES (1E2);
INSERT INTO E091_01 VALUES (1E2);
INSERT INTO E091_01 VALUES (1E2);

SELECT AVG(S1) FROM E091_01;
-- MySQL Result: 0.0099999997764826
-- DB2 Result: +1.00000000000000E002
```

The "implementation defined" loophole!

64.1.1 Text:

I'll show now what happens with the average function. The answer here should be zero point zero one, and MySQL comes close to that, but DB2 comes closer. Does that mean that MySQL violates the Standard? No, it does not mean that.

For matters such as how accurate the result of a floatingpoint arithmetic calculation must be, the Standard says that we, the implementors, may define the requirement to suit ourselves.

This is the quote implementation defined unquote loophole. It appears frequently in the Standard, for all those minor matters that might depend on what has happened in the past, or depend on what the underlying C library has, or result in a difference that's too small to worry about. The result is that two DBMSs can follow the Standard but still can return different results, as happens here.

65 Slide 65

65.1 Progress: Small Things that Failed Before, and Work Now

```
1. SELECT COUNT(ALL xxx) FROM t;
2. INSERT INTO t SELECT MAX(s1) FROM t;
3. CREATE TABLE t (s1 INT, UNIQUE (s1));
   ALTER TABLE t DROP PRIMARY KEY;
4. CREATE TABLE t (s1 INT, PRIMARY KEY(s1));
5. SELECT LOCALTIME;
6. UPDATE t SET s1 = DEFAULT;
```

65.1.1 Text:

Incremental improvements again.

- One. At the start of the period, MySQL didn't accept the word ALL, which does nothing, in a set function. Now we support this syntax, as required by the SQL Standard.
- Two. At the start of the period, MySQL didn't support use of the same table name in the SELECT as was being used in the INSERT. Now, this is supported. By the way, the restriction that we had here was in fact conformant with an older version of the SQL Standard.
- Three. At the start of the period, we treated the first unique column of a table as a primary key. Now we don't.
- Four. At the start of the period, MySQL didn't allow the primary key clause to be used on a column that wasn't explicitly declared as NOT NULL. This is now allowed, as required by standard SQL.
- Five. At the start of the period, MySQL didn't support the standard SQL LOCALTIME function. In fact, we didn't have any support for time zones at all. Now we do.
- Six. At the start of the period, we didn't allow you to use the word DEFAULT in an UPDATE statement. Now we do.

So, six little bits of progress. Just to show that we slog along with the minor stuff. Not everything is a headline.

66 Slide 66

66.1 The ANSI Mode

```
mysql> ansi
mysql> set sql_mode = 'ansi';
```

Advantages:

```
select 'a' || 'b' ... returns 'ab'
REAL is a lowprecision float
"x" is a legitimate delimited identifier
```

Disadvantages:

```
where 'a' || 'b' = 'ab' ... fails
SHOW CREATE TABLE doesn't show all clauses
```

66.1.1 Text:

Should lovers of ANSI (or standard) SQL use MySQL's dash dash ANSI switch all the time? That is, should they start the server with the dash dash ansi switch, or maybe should they change the sql mode by saying set sql_mode equals ansi?

The advantages to doing so, as I'm showing on this slide, are that the standard SQL concatenation operator (the two vertical bars) works, and that the REAL data type is what it should be, and that you can use double quote marks instead of back ticks to delimit names.

But unfortunately, the precedence of the concatenation operator is wrong, so conditional expressions won't work unless you use parentheses. And statements like SHOW CREATE TABLE won't be informative any more.

I myself don't always use the ansi mode. Its advantages are trivial.

67 Slide 67

67.1 Major Missing Features (NonCore)

Triggers
UDTs
XML (SQL:2003)

67.1.1 Text:

Now, to counter the nonCore features that we do have, I have to mention that there are many nonCore features that MySQL doesn't support. These are the ones that people ask about most frequently.

68 Slide 68

68.1 New Options

MaxDB (formerly SAPDB)
Cluster (formerly ndb cluster)

68.1.1 Text:

At MySQL, we have another street that we can go down, if we want to allow for some standard SQL features that the basic MySQL package, or the basic MySQL package with InnoDB, doesn't have.

First, we can point out that MySQL AB also supplies MaxDB, which has quite a few extra features, like views and stored procedures.

We can also point out that there is a new storage engine called the cluster storage engine, formerly known as ndb cluster. It allows for some transactional control that isn't available with the MyISAM storage engine.

I'm just suggesting that some of the features that some people might want today are available today from MySQL AB, but this is something that I only suggest. When I say that MySQL supports or is going to support or does not support something, I'm always referring to the main line product that most of you know best.

69 Slide 69

69.1 Contributions by MySQL Employees

1. <modulus expression> ... found by Sergei Golubchik
2. NULLS FIRST / LAST ... found by Peter Gulutzan

69.1.1 Text:

What about our own participation in the Standards development process? MySQL AB isn't a member of the ANSI or ISO Standards committees, but our employees have contributed in small ways.

On the slide here, I mention two bugs in the Standard that MySQL employees found and reported. The first was a mistake in the description of the MOD expression, found by our man in Osnabrück, Sergei Golubchik. The second was an ambiguity regarding null ordering, found by me.

So we try to do our part.

70 Slide 70

70.1 Final Mark

Much Improved Since Version 3.23	YES
Supports All but 4 Major NonCore Features	YES
Catching up to Classmates in Higher Grades	YES

70.1.1 Text:

Here is the final mark on my progress report for MySQL and standard SQL.

Little MySQL is doing very well and pays attention in class. Must complete some requirements before graduation in version 6. Still behind some classmates but is catching up quickly. And those classmates are a lot older, so really, little MySQL's parents should be proud.