MariaDB Course

Conducted by:
Mohammed Abdul Sami
Contact: sami.ma@gmail.com

Introduction	5
About Author	5
RDBMS Concepts	6
RDBMS Terms	6
E. F. Codd's Rules for RDBMS	6
MariaDB	9
Architecture	9
Connection Manager	9
SQL Interface	10
Parser	10
Optimizer	10
Caches & Buffers	10
Storage Engines	10
Three layer model	10
Client / Server Connection phase	11
Protocols	12
Storage Engines in MariaDB	13
InnoDB	14
Key Features	14
Innodb Multi Versionning	15
Innodb Locks	15
MyISAM	16
Key Features	16
MEMORY Storage Engine	17
Key Features	17
CSV Storage Engine	18
MERGE storage engine	18
Aria storage engine	18
Sequence Storage Engine	19
Data Types	19
Numeric Data Type	19
Integers	19
Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC	20

Floating-Point Types (Approximate Value) - FLOAT, DOUBLE 20
Bit-Value Type - BIT 21
Date & Time Data Type 21
Date: 21
DateTime 21
Timestamp 22
Time: 22
YEAR 22
String Types 22
Char : 22
Varchar: 22
The BLOB and TEXT Types 22
The ENUM Type 23
Set data type 23
SQL Language 23
MariaDB Functions 24
Transactions 25
Transactions and Isolation Levels 25
Transaction Problems 25
ACID Properties 26
InnoDB Locking 27
Innodb Locks 27
Table Partitioning 28
Why to Use Table Partition 28
Types of Partitioning 28
Range Partition 28
List Partition 29
HASH Partition 29
Key Partitioning 30
InnoDB Indexes 30
Primary Key 30
Unique Key 31
Non-Unique indexes (Plain Indexes) 31
Full-Text Indexes 32

Index Extensions	32
Displaying Indexes	32
Rebuilding Indexes	33
Index Statistics	33
MariaDB Programming	34
Table Optimization & Statistics	42
Analyze Table	42
Optimize Table	42
Query Optimization (Explain)	43
Explain	43
Columns EXPLAIN output	44
Columns to watch from EXPLAIN output and Expected Values	45
Normalization	46
Database Anomalies	46
Normalization	46
Un-Normalized Form (UNF)	46
First Normal Form (1NF)	46
Second Normal Form (2NF)	47
Third Normal Form (3NF)	47
Fourth Normal Form (4NF)	48

Introduction

This Document is created as reference material for MariaDB training conducted at DBS Bank, Hyderabad.

About Author

Mohammed Abdul Sami is an experienced MySQL DBA & System Administrator having more than 19 years of experience in managing Database Servers and Windows / Unix / Linux Servers.

Since 2014 he is mostly involved in trainings and a regular instructor at Oracle University for MySQL courses.

RDBMS Concepts

RDBMS Terms

RDBMS stands for relational database management system. A relational model can be represented as a table of rows and columns. A relational database has following major components:

1. Table or Relation

In Relational database model, a table is a collection of data elements organised in terms of rows and columns. A table is also considered as a convenient representation of relations. But a table can have duplicate row of data while a true relation cannot have duplicate data.

2. Record or Tuple or Row

A single entry in a table is called a Tuple or Record or Row. A tuple in a table represents a set of related data.

3. Field or Column name or Attribute

A table consists of several records(row), each record can be broken down into several smaller parts of data known as Attributes.

4. Domain or Datatype

When an attribute is defined in a relation(table), it is defined to hold only a certain type of values, which is known as Attribute Domain.

5. Instance

The data stored in database at a particular moment of time is called instance of database. .

6. Schema

A relation schema describes the structure of the relation, with the name of the relation(name of table), its attributes and their names and type.

7. Keys

A relation key is an attribute which can uniquely identify a particular tuple(row) in a relation(table).

E. F. Codd's Rules for RDBMS

Edgar Frank Codd was an English computer scientist who, while working for IBM, invented the relational model for database management, the theoretical basis for relational databases.

Rule 1 – Information Rule :-
All information is explicitly and logically represented in exactly one way – by data values in tables. If an item of data does not reside somewhere in a table in the database, then it does not exist.

Rule 2 – Rule of Guaranteed Access :-

Every item of data must be logically addressable by resorting to a combination of table name, primary key value and a column name.

Rule 3 – The Systematic Treatment of Null Values :-

The RDBMS handles records that have unknown or inapplicable values in a pre-defined fashion.
The RDBMS distinguishes between zeros, blanks and nulls in the records hand handles such values in a consistent manner that produces correct answers, comparisons and calculations. Through the set of rules for handling nulls, users can distinguish results of the queries that involve nulls, zeros and blanks.

Rule 4 – The Database Description Rule :-

The description of a database and in its contents are database tables and therefore can be queried on-line via the data manipulation language.

There must be a Data Dictionary within the RDBMS that is constructed of tables and/or views that can be examined using SQL.

Rule 5 – The Comprehensive Sub-language rule :-

A RDBMS may support several languages. But at least one of them should allow user to do all of the following:

- Define tables and views
- Query and update the data
- Set integrity constraints
- Set authorizations
- Define transactions.

RDBMS must be completely manageable through its own dialect of SQL.

Rule 6 – The View Updating Rule :-

All views that can be updated in theory, can also be updated by the system.
Data consistency is ensured since the changes made in the view are transmitted to the base table and vice-versa.

Rule 7 – High Level Insert, Update and Delete Rule

The RDBMS supports insertions, updation and deletion at a table level.
This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.

Rule 8 – The Physical Independence Rule :-

The execution of adhoc requests and application programs is not affected by changes in the physical data access and storage methods.
Database administrators can make changes to the physical access and storage method which improve performance and do not require changes in the application programs or requests. Here the user specified what he wants an need not worry about how the data is obtained.

Rule 9 – Logical Data Interdependence Rule:-

Logical changes in tables and views such adding/deleting columns or changing fields lengths need not necessitate modifications in the programs or in the format of adhoc requests.
The database can change and grow to reflect changes in reality without requiring the user intervention or changes in the applications. For example, adding attribute or column to the base table should not disrupt the programs or the interactive command that have no use for the new attribute.

Rule 10 – Integrity Independence Rule :-

Integrity constraints must be specified separately from application programs and stored in the catalog. It must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications.
The following Integrity rules(Constraints) should apply to every RDBMS :-

- No component of a primary key can have missing values – this is the basic rule of Entity Integrity.
- For each distinct foreign key value there must exist a matching primary key value in the same domain. Conformation to this rule ensures what is called Referential integrity.

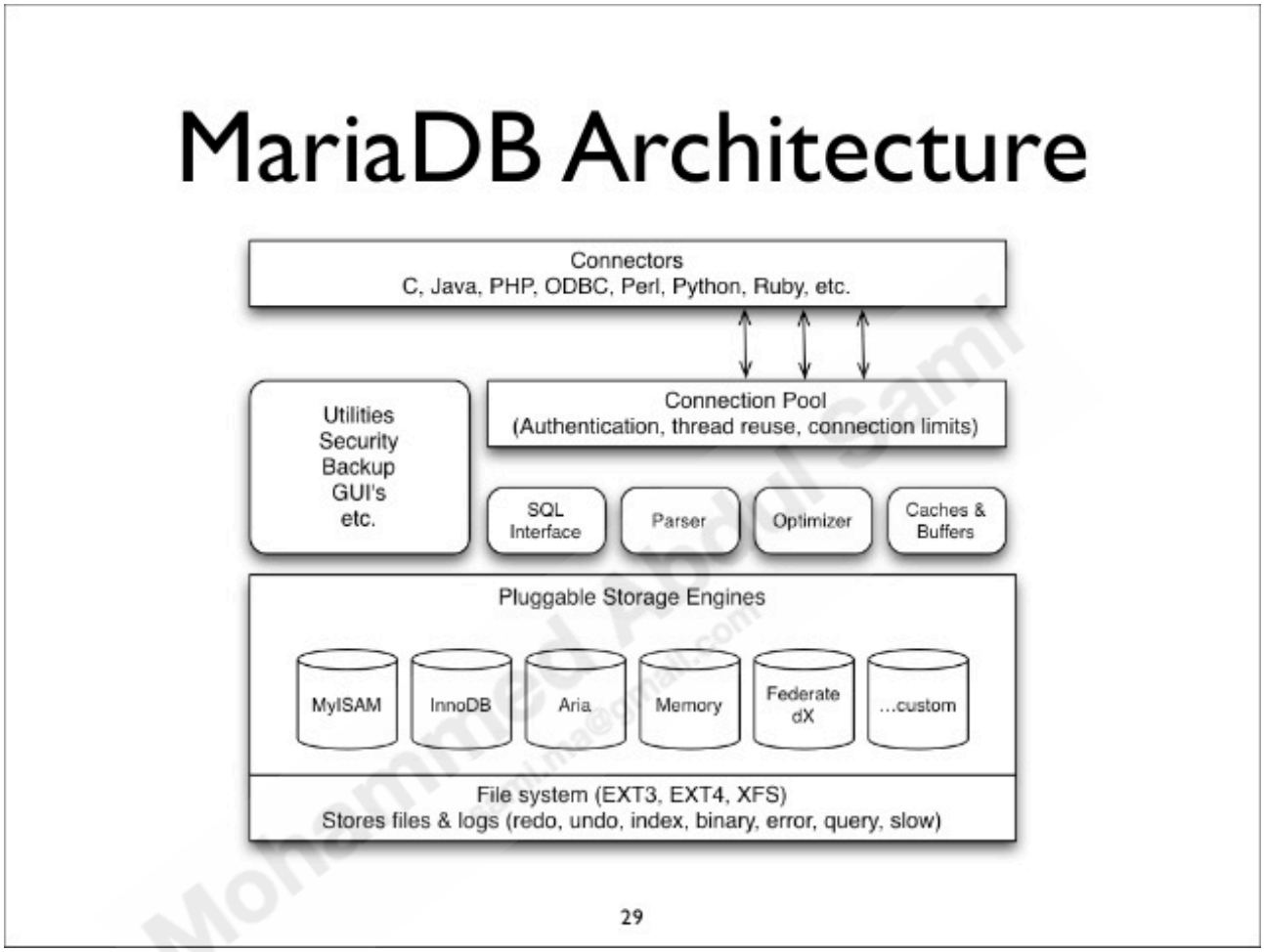
Rule 11 – Distribution Rule :-

Application programs and adhoc requests are not affected by change in the distribution of physical data.
Improved systems reliability since application programs will work even if the programs and data are moved in different sites.

Rule 12 – No Subversion Rule :-

If the RDBMS has a language that accesses the information of a record at a time, this language should not be used to bypass the integrity constraints.

MariaDB
Architecture



Connection Manager

Connection manager threads handle client connection requests on the network interfaces that the server listens to, Unix / Linux sockets, Windows Shared Memory connections and windows named pipes. User credentials are validated before establishing connection. Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection.

A new thread is created when necessary but avoids doing so by consulting the thread cache first to see whether it contains a thread that can be reused for the connection.

When a connection ends, its thread is returned to the thread cache.

SQL Interface

SQL interface provides mechanism to receive commands and transmit results back to user. It is designed to accept ANSI standard SQL queries but some of queries may use MySQL specific syntax.

Parser

When a client issues a query, a new thread is created and the SQL statement is forwarded to the parser for syntactic validation (or rejection due to errors).

Optimizer

These strategies may include rewrite request, make sure to read the order of thetable, decide which indexes

Caches & Buffers

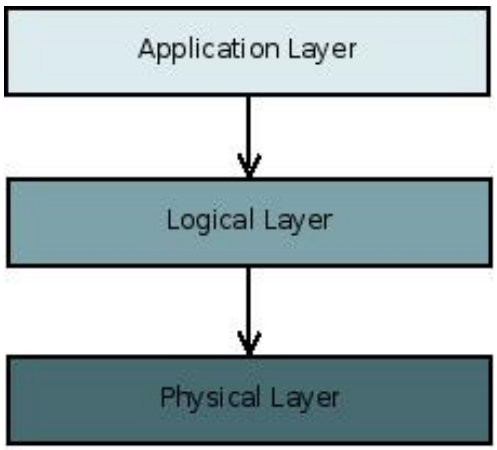
Caches & Buffers subsystem is responsible for ensuring most frequently used data is available in most efficient manner. Caches dramatically increase the response time for such data as that is available in memory eliminating the need to read from disk.

Storage Engines

This is one of the distinct feature of MySQL which allows to tune database performance by selecting suitable storage engine for their application.

Storage engines responsible for the physical storage and retrieval of data from disk files.

Three layer model



- The application layer contains common network services for connection handling, authentication and security. This layer is where different clients interact with MySQL these clients can be written in different API's:.NET, Java, C, C++, PHP, Python, Ruby, Tcl, Eiffel, etc...
- The Logical Layer is where the MySQL intelligence resides, it includes functionality for query parsing, analysis, caching and all built-in functions (math, date...). This layer also provides functionality common across the storage engines.
- The Physical Layer is responsible for storing and retrieving all data stored in “MySQL”. Associated with this layer are the storage engines, which MySQL interacts with very basic standard API's. Each storage engine has it strengths and weakness, some of this engines are MyISAM, InnoDB, CSV, NDB Cluster, Falcon, etc...

Client / Server Connection phase

The Connection Phase performs these tasks:

- exchange the capabilities of client and server
- setup SSL communication channel if requested
- authenticate the client against the serve

It starts with the client connecting to the server which may send a ERR packet and finish the handshake or send a Initial Handshake Packet which the client answers with a Handshake Response Packet. At this stage client can request SSL connection, in which case an SSL communication channel is established before client sends its authentication response.

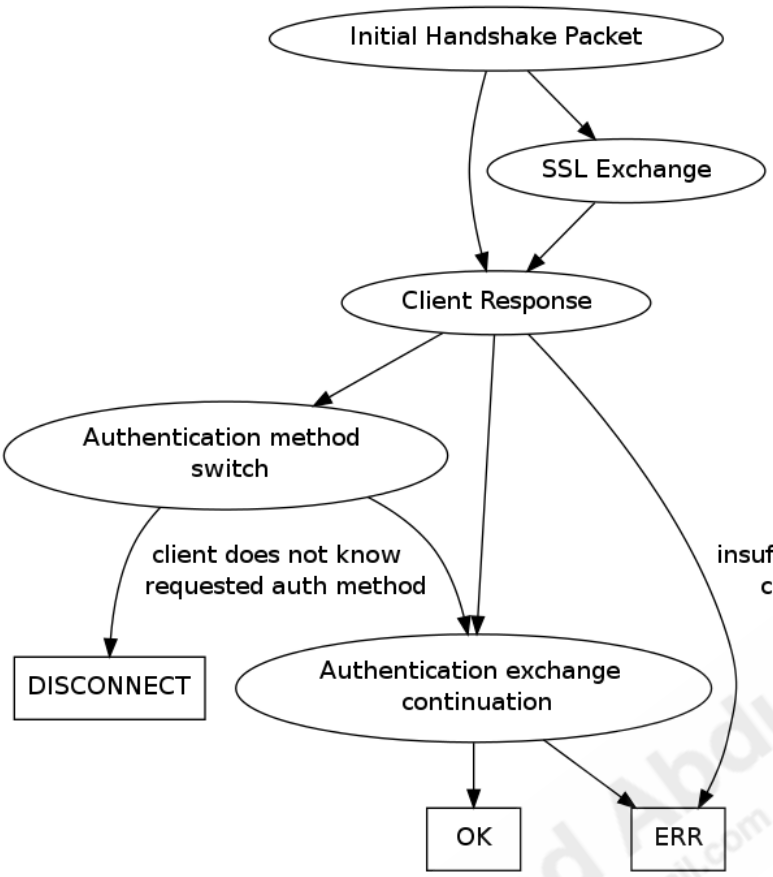


Figure 2 - Client Authentication flow

Protocols

Below are the main protocols which are used by client to connect with MySQL server.

- TCP/IP
- Unix socket file
- Named pipe
- Shared memory

TCP/IP connections are supported by any MySQL server unless the server is started with the –skip-networking option. These connections are supported by all type of Operating Systems and also connect from locally or remotely.

Unix socket file connections are supported by all Unix servers and this can be connect from local only.

Named-pipe connections are supported only by Windows servers and are disabled by default. To enable named-pipe connections, you have to start the mysql-nt server with the –enable-named-pipe option.

Shared-memory connections are supported by all Windows servers and are disabled by default. To enable shared-memory connections, you have to start the server with the –shared-memory option.

Storage Engines in MariaDB

A storage engine is a software module that a database management system uses to create, read, update data from a database. There are two types of storage engines in MySQL. Transactional and non-transactional.

Transactional Storage Engine: A MySQL storage Engine which supports DML Transactions.

Non-Transsctional Storage Engine: A MySQL storage Engine which presents data as is and does not support tractions.

MySQL supports several storage engines that act as handlers for different table types. MySQL storage engines include both those that handle transaction-safe tables and those that handle non-transaction-safe tables.

Run following command to list out all the supported storage engines in your MariaDB installation.

```
SHOW ENGINES\G
```

Some of the supported storage engines

- MyISAM
- InnoDB
- Memory
- CSV
- Merge
- Aria
- Sequence

InnoDB

InnoDB is a high-reliability and high-performance transactional storage engine for MySQL. Starting with MySQL 5.5, it is the default MySQL storage engine.

Key Features

- Its design follows the ACID model, with transactions featuring commit, rollback, and crash-recovery capabilities to protect user data.
- Row-level locking and Oracle-style consistent reads increase multi-user concurrency and performance.
- InnoDB tables arrange your data on disk to optimize common queries based on primary keys. Each InnoDB table has a primary key index called the clustered index that organizes the data to minimize I/O for primary key lookups.
- To maintain data integrity, InnoDB also supports FOREIGN KEY referential-integrity constraints.
- You can freely mix InnoDB tables with tables from other MySQL storage engines, even within the same statement. For example, you can use a join operation to combine data from InnoDB and MEMORY tables in a single query.
- To determine whether your server supports InnoDB use the SHOW ENGINES statement.

Storage limits	64TB	**Transactions**	Yes	**Locking granularity**	Row
MVCC	Yes	**Geospatial data type support**	Yes	**Geospatial indexing support**	No

B-tree indexes	Yes	**T-tree indexes**	No	**Hash indexes**	No[a]
Full-text search indexes	Yes[b]	**Clustered indexes**	Yes	**Data caches**	Yes
Index caches	Yes	**Compressed data**	Yes[c]	**Encrypted data[d]**	Yes
Cluster database support	No	**Replication support[e]**	Yes	**Foreign key support**	Yes
Backup / point-in-time recovery[f]	Yes	**Query cache support**	Yes	**Update statistics for data dictionary**	Yes

-

Table 1 - Innodb Features

Innodb Multi Versionning

- Supports Concurrency and Rollback
- Rollback segment contains data of concurrency & Rollback stored in Tablespace
- 3 fields added to each row
    - 6 byte DB_TRX_ID - transaction identifier of last transation
    - 7 byte DB_ROLL_PTR - points to rollback segment
    - 6 byte DB_ROW_ID - row id of each row
- A special bit column used to mark the deleted records
- Undo logs are of Inserts & updates types
- Insert undo logs discarded as soon as trx committed
- Update undo logs discarded when all snapshots for consistent read are released

Innodb Locks

- A shared (S) lock permits the transaction that holds the lock to read a row.
- An exclusive (X) lock permits the transaction that holds the lock to update or delete a row.

Intention Locks

- Intention shared (IS): Transaction T intends to set S locks on individual rows in table t.
- Intention exclusive (IX): Transaction T intends to set X locks on those rows.

InnoDB Lock compitability Chart

	X	***IX***	***S***	***IS***
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

MyISAM

Before MySQL 5.5.5, **MyISAM** is the default storage engine. (The default was changed to **InnoDB** in MySQL 5.5.5.)

MyISAM is based on the older (and no longer available) **ISAM** storage engine but has many useful extensions

Key Features

Storage limits	256TB	**Transactions**	No	**Locking granularity**	Table
MVCC	No	**Geospatial data type support**	Yes	**Geospatial indexing support**	Yes
B-tree indexes	Yes	**T-tree indexes**	No	**Hash indexes**	No
Full-text search indexes	Yes	**Clustered indexes**	No	**Data caches**	No
Index caches	Yes	**Compressed data**	Yes	**Encrypted data**	Yes
Cluster database support	No	**Replication support**	Yes	**Foreign key support**	No

Backup / point-in-time recovery	Yes	**Query cache support**	Yes	**Update statistics for data dictionary**	Yes

Table 2 - MyISAM Features

Each MyISAM table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. A .frm (MySQL v8 onwards .frm files no more store metadata) file stores the table format. The data file has an .MYD (MYData) extension. The index file has an .MYI (MYIndex) extension

MEMORY Storage Engine

The MEMORY storage engine (formerly known as HEAP) creates special-purpose tables with contents that are stored in memory. Because the data is vulnerable to crashes, hardware issues, or power outages, only use these tables as temporary work areas or read-only caches for data pulled from other tables.

Key Features

Storage limits	RAM	Transactions	No	Locking granularity	Table
MVCC	No	Geospatial data type support	No	Geospatial indexing support	No
B-tree indexes	Yes	T-tree indexes	No	Hash indexes	Yes
Full-text search indexes	No	Clustered indexes	No	Data caches	N/A
Index caches	N/A	Compressed data	No	Encrypted data	Yes
Cluster database support	No	Replication support	Yes	Foreign key support	No

Backup / point-in-time recovery	Yes	Query cache support	Yes	Update statistics for data dictionary	Y

Table 3 - Memory Storage Engine Features

CSV Storage Engine

The CSV storage engine stores data in text files using comma-separated values format. The CSV storage engine is always compiled into the MySQL server.

When you create a CSV table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. The storage engine also creates a data file. Its name begins with the table name and has a .CSV extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

- The CSV storage engine does not support indexing.
- Partitioning is not supported for tables using the CSV storage engine.
- All tables that you create using the CSV storage engine must have the NOT NULL attribute on all columns.

MERGE storage engine

The MERGE storage engine, also known as the MRG_MyISAM engine, is a collection of identical MyISAM tables that can be used as one big table.

“Identical” means that all tables have identical column and index information.

An alternative to a MERGE table is a partitioned table, which stores partitions of a single table in separate files.

Aria storage engine

Aria is a new storage engine for MySQL and MariaDB which was developed with the goal of being the default transactional AND non-transactional storage engine for MariaDB and MySQL.

It is a crash-safe alternative to MyISAM. It is not yet transactional but plans to add proper support for database transactions at some point in the future.

It is being actively developed by the same core MySQL engineers who developed the MySQL server and the MyISAM, MERGE, and MEMORY storage engines.

Sequence Storage Engine

A Sequence engine allows the creation of ascending or descending sequences of numbers (positive integers) with a given starting value, ending value and increment.

It creates completely virtual, ephemeral tables automatically when you need them. There is no way to create a Sequence table explicitly. Nor are they ever written to disk or create .frm files.

Usage:

```
MariaDB [d1]> select * from seq_1_to_5_step_2
```


Data Types

Data type defines what type data will be stored in a particular column.

- Numeric Data Types
- Date and Time Data Types
- String Data Types

Numeric Data Type

Attributes: ZEROFILL, UNSIGNED, Auto_Increment and not null

Integers

Type	Storage (Bytes)	Minimum Value (Signed/Unsigned)	Maximum Value (Signed/Unsigned)
TINYINT	1	-128	127

		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC

The DECIMAL and NUMERIC types store exact numeric data values

- Used to storing monetary data
- Preserve exact precision
- Max number of digits are 65
- Default is 10 digits

Floating-Point Types (Approximate Value) - FLOAT, DOUBLE

- MySQL perform rounding while storing values
- Precision is supported
- Precision is only used to determine storage size
- Precision upto 23 results in 4 byte storage
- Precision from 23 to 53 results in 8 byte (DOUBLE)
- FLOAT handling may vary with H/W architecture

Range:

Float

-3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38

Double

1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.

Bit-Value Type - BIT

- Stores data in bits [binary format]
- Width can be 1 to 64
- Value can be assigned by b'value'
- Can be comparison is done as (1/0), 'true' or b'value'

Date & Time Data Type

Data Type	Storage Required
Date	3 bytes
TIME	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
YEAR	1 byte

Date:

- Only date is stored and time is omitted
- Range The supported range is '1000-01-01' to '9999-12-31'
- Format 'YYYY-MM-DD'

DateTime

- Contain both date and time

- Range '1000-01-01 00:00:00' to'9999-12-31 23:59:59'
- Format 'YYYY-MM-DD HH:MM:SS'

Timestamp

- Contain both date and time part
- Range '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC
- Format 'YYYY-MM-DD HH:MM:SS‘
- Converts date&time to UTC and stores while retrieving converts from UTC to current time zone.
- When using on update clause it will change it to current timestamp for all changes in that row

Time:

- Stores time data
- Range '-838:59:59' to '838:59:59‘
- Format 'HH:MM:SS'

YEAR

- Stores year data
- Range: '1901' to '2155‘
- Format ‘yyyy’

String Types

Char :

- Fixed length. White spaces are padded to left.
- Length 0 to 255
- Storage : each character takes a byte

Varchar:

- Variable length. Length is specified while declaration
- Length can be upto 65,535
- Storage : upto 255 1 + Length and beyond 255 2 + Length

Storage differs depending the character set used for example latin1 used a byte per character and utf8 uses 3 bytes per character.

The BLOB and TEXT Types

Data Type	Storage Required
TINYBLOB, TINYTEXT	*L* + 1 bytes, where *L* < 2^8

BLOB, TEXT	*L* + 2 bytes, where *L* < 2^{16}
MEDIUMBLOB, MEDIUMTEXT	*L* + 3 bytes, where *L* < 2^{24}
LONGBLOB, LONGTEXT	*L* + 4 bytes, where *L* < 2^{32}

The ENUM Type

- List of permitted values declared at time of table creation
- String inputted are encoded in numbers
- 65,535 distinct elements
- 1 or 2 bytes storage

Set data type

- List of permitted values
- A column can have one or more values from the list
- Elements in set should not contain ,
- Each element value appears once when retrieved. Irrespective of how many times repeated in a column
- 64 distinct members
- Upto 8 bytes storage

SQL Language

DDL : Data Definition Language allows to create and manage database objects in RDBMS. It consists of create, alter and drop commands

CREATE DATABASE is used to create a new, empty database.
DROP DATABASE is used to completely destroy an existing database.
USE is used to select a default database.
CREATE TABLE is used to create a new table, which is where your data is actually stored.
ALTER TABLE is used to modify an existing table's definition.
DROP TABLE is used to completely destroy an existing table.
DESCRIBE shows the structure of a table.

DML : Data Manipulation Language
SELECT is used when you want to read (or select) your data.
INSERT is used when you want to add (or insert) new data.
UPDATE is used when you want to change (or update) existing data.
DELETE is used when you want to remove (or delete) existing data.
TRUNCATE is used when you want to empty (or delete) all data from the template.

TCL - Transaction Control Language
START TRANSACTION is used to begin a transaction.
COMMIT is used to apply changes and end transaction.
ROLLBACK is used to discard changes and end transaction.

MariaDB Functions

String Function

Function	Description
ASCII	Returns the ASCII value for the specific character
CHAR_LENGTH	Returns the length of a string (in characters)
CONCAT	Adds two or more expressions together
LCASE	Converts a string to lower-case
LEFT	Extracts a number of characters from a string (starting from left)
LENGTH	Returns the length of a string (in bytes)
LOWER	Converts a string to lower-case
POSITION	Returns the position of the first occurrence of a substring in a string
REPEAT	Repeats a string as many times as specified
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting from right)
SUBSTR	Extracts a substring from a string (starting at any position)
UCASE	Converts a string to upper-case
UPPER	Converts a string to upper-case

Numeric Functions

Function	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
ATAN	Returns the arc tangent of one or two numbers
CEILING	Returns the smallest integer value that is >= to a number
COS	Returns the cosine of a number
COT	Returns the cotangent of a number
LN	Returns the natural logarithm of a number
LOG	Returns the natural logarithm of a number
LOG10	Returns the natural logarithm of a number to base 10
LOG2	Returns the natural logarithm of a number to base 2
MOD	Returns the remainder of a number divided by another number
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of another number
POWER	Returns the value of a number raised to the power of another number
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places
SIGN	Returns the sign of a number
SIN	Returns the sine of a number
SQRT	Returns the square root of a number
TAN	Returns the tangent of a number

TRUNCATE	Truncates a number to the specified number of decimal places

Aggregate Functions

Function	Description
SUM	Calculates the sum of a set of values
AVG	Returns the average value of an expression
MAX	Returns biggest values in given set
MIN	Returns least value in given set
COUNT	Count number of items in given set

Date Functions

Function	Description
ADDDATE	Adds a time/date interval to a date and then returns the date
ADDTIME	Adds a time interval to a time and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns the date
DATE_SUB	Subtracts a time/date interval from a date and then returns the date
DAY	Returns the day of the month for a given date
DAYNAME	Returns the weekday name for a given date
DAYOFWEEK	Returns the weekday index for a given date
DAYOFYEAR	Returns the day of the year for a given date
MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
NOW	Returns the current date and time
YEAR	Returns the year part for a given date

Transactions

Transactions and Isolation Levels

In databases, a transaction is a set of separate actions that must all be completely processed, or none processed at all. When you think of a transaction, you should think of the phrase “all or nothing”. Transactions are completed by COMMIT or ROLLBACK SQL statements.

Transaction Problems

Three common problems with Concurrent transactions:

1. **Dirty Read :** When a transaction reads the changes made by another uncommitted transaction

2. **Non-repeatable read :** When changes from another committed transaction cause a prior read operation to be non-repeatable

3. **Phantom read (or phantom row) :** A row that appears but was not previously visible within the same transaction

Four isolation levels:

1. **READ UNCOMMITTED :** Allows a transaction to see uncommitted changes made by other transactions
2. **READ COMMITTED :** Allows a transaction to see committed changes made by other transactions
3. **REPEATABLE READ :** Ensures consistent SELECT output for each transaction Default level for indoor
4. **SERIALIZABLE :** Completely isolates the effects of a transaction from others

Isolation Level Problems Chart for InnoDB

Isolation Level	Dirty Reads	Non Repeatable Read	Phantom Rows
Read Uncommitted	YES	YES	YES
Read Committed	NO	YES	YES
Repeatable Read	NO	NO	NO (For Other RDBMS it is YES)
Serializable	NO	NO	NO

Transaction Control Statements supported in MariaDB

- START TRANSACTION (or BEGIN): Explicitly begins a new transaction
- SAVEPOINT: Assigns a location in the process of a transaction for future reference
- COMMIT: Makes the changes from the current transaction permanent
- ROLLBACK: Cancels the changes from the current transaction
- ROLLBACK TO SAVEPOINT: Cancels the changes executed after the savepoint
- RELEASE SAVEPOINT: Removes the savepoint identifier

ACID Properties

1. **Atomic :** All statements execute successfully or are canceled as a unit.
2. **Consistent :** A database that is in a consistent state when a transaction begins is left in a consistent state by the transaction.
3. **Isolated :** One transaction does not affect another.

4. **Durable :** All changes made by transactions that complete successfully are recorded properly in the database. No changes are lost

InnoDB Locking

InnoDB implements standard row-level locking where there are two types of locks, shared (S) locks and exclusive (X) locks.

Innodb Locks

- A shared (S) lock permits the transaction that holds the lock to read a row. Prevents other transactions from changing locked rows.
- An exclusive (X) lock permits the transaction that holds the lock to update or delete a row. While preventing other transactions from reading or writing to the locked row(s).

Intention Locks

Intention locks are table-level locks that indicate which type of lock (shared or exclusive) a transaction requires later for a row in a table. There are two types of intention locks:

- Intention shared (IS): Transaction T intends to set S locks on individual rows in table t.
- Intention exclusive (IX): Transaction T intends to set X locks on those rows.

Acquiring InnoDB intention locks:

```
SELECT ... LOCK IN SHARE MODE; -- sets
an IS lock

SELECT ... FOR UPDATE; --  sets an IX lock.
```

InnoDB Lock compatibility Chart

	X	***IX***	***S***	***IS***
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible

IS	Conflict	Compatible	Compatible	Compatible

InnoDB Dead Locks

InnoDB automatically detects transaction deadlocks and rolls back a transaction or transactions to break the deadlock. InnoDB tries to pick small transactions to roll back, where the size of a transaction is determined by the number of rows inserted, updated, or deleted.

Table Partitioning

Partitioning allows you to store parts of your table in their own logical & physical space. With partitioning, you want to divide up your rows based on how you access them. If you partition your rows and you are still hitting all the partitions, it does you no good. The goal is that when you query, you will only have to look at a subset of the data to get a result, and not the whole table

Why to Use Table Partition

- Partitioning makes it possible to store more data in one table than can be held on a single disk
- Data that loses its usefulness can often be easily removed from a partitioned table by dropping the partition (or partitions) containing only that data.
- Some queries can be greatly optimized in virtue of the fact that data satisfying a given WHERE clause can be stored only on one or more partitions, which automatically excluding any remaining partitions from the search.

Types of Partitioning

- List Partitions
- Range Partitions
- HASH Partitioning
- Key Partitioning

Range Partition

This type of partitioning assigns rows to partitions based on column values falling within a given range.

Creating RANGE Partition

```
mysql> CREATE TABLE <table name> (
```

```
column def …)
PARTITION BY RANGE (column name)
(
PARTITION p0 VALUES LESS THAN (values),
PARTITION p1 VALUES LESS THAN (values),
…
PARTITION p1 VALUES LESS THAN MAXVALUE ) ;
```

List Partition

Similar to partitioning by RANGE, except that the partition is selected based on columns matching one of a set of discrete values.

Creating List Partition

```
mysql> CREATE TABLE <table name> (
column def …)
PARTITION BY LIST (column name)
(
PARTITION p0 VALUES IN  (List of Values),
PARTITION p1 VALUES IN  (List of Values),
…
PARTITION p1 VALUES IN  (List of Values) ;
```

HASH Partition

With this type of partitioning, a partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table.

Creating HASH Partition

```
mysql> CREATE TABLE <table name> (
column def …)
PARTITION BY HASH (column name)
PARTITIONS <Number of Partitions>;
```

Key Partitioning

This type of partitioning is similar to partitioning by HASH, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function.

Creating Key Partition

```
mysql> CREATE TABLE <table name> (

column def …)

PARTITION BY KEY (column name)

PARTITIONS <Number of Partitions>;
```

InnoDB Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. This is much faster than reading every row sequentially.

There are 4 kinds of indexes

1. Primary Key
2. Unique Key
3. Index (non-unique)
4. Full-Text Index

Primary Key

A primary key is unique and can never be null. It will always identify only one record, and each record must be represented. Each table can only have one primary key.
In XtraDB/InnoDB tables the table data is arranged in the order of primary key (clustered index) If a primary key does it will look for suitable unique key to be used as clustered index and if no unique is present then it will add an invisible column to be used and clustered index.

Adding Primary Key

```
CREATE TABLE <table name> (....., PRIMARY
KEY(column, [column2]);

ALTER TABLE <table name>  ADD PRIMARY KEY
(column, [column2])
```

Note: Primary Key cannot be added with CREATE INDEX statement.

Unique Key

A Unique Index must be unique, but it can be null. So each key value identifies only one record, but not each record needs to be represented.

Adding Unique Key

```
CREATE TABLE <table name> (....., UNIQUE
KEY(column, [column2]);

ALTER TABLE <table name>  ADD UNIQUE (column);

CREATE UNIQUE INDEX <index name> ON <table
name> (column, [column2]);
```

Non-Unique indexes (Plain Indexes)

These indexes can be created on any column of table which may contain non-unique values.

Adding Plain Indexes

```
CREATE TABLE <table name> (....., INDEX
(column, [column2]);

ALTER TABLE <table name>  ADD INDEX (column,
[column2]);

CREATE INDEX <index name> ON <table name>
(column, [column2]);
```

Full-Text Indexes

FULLTEXT indexes are used for full-text searches. FULLTEXT indexes are supported only for CHAR, VARCHAR and TEXT columns. Indexing always takes place over the entire column and column prefix indexing is not supported.

```
CREATE TABLE <table name> (....., FULLTEXT
(column);

ALTER TABLE <table name>  ADD FULLTEXT
(column);

CREATE FULLTEXT INDEX <index name> ON <table
name> (column);
```

Index Extensions

InnoDB automatically extends each secondary index by appending the primary key columns to it. This improves the performance of secondary (non unique) indexes.

Displaying Indexes

All created indexes information can be queried using SHOW INDEXES FROM or Querying STATISTICS table from information_schema database.

```
SHOW INDEXES FROM <table name>\G

SELECT DISTINCT TABLE_NAME, INDEX_NAME FROM
INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = '<database name';
```

Rebuilding Indexes

Rebuilding indexes can improve performance. Following commands rebuilds Indexes

```
ALTER TABLE <table name> FORCE;

OPTIMIZE TABLE <Table Name>;
```

Index Statistics

Innodb by default stores the statistics persistently in mysql.innodb_index_stats table.

Innodb recalculates the statistics automatically when 10% rows of any tables changes. It samples 20 pages by default to recalculating statistics.
Optionally after executing metadata statements such as SHOW TABLE STATUS, or when querying INFORMATION_SCHEMA.TABLES statistics are recalculated

MariaDB Programming

Stored Procedures

Stored procedures are database program which can b used to manipulate data programmatically on a mariadb server. Once a procedure is created it is saved on server and can be executed any number of times.

Syntax:

CREATE PROCEDURE procedure_name [(parameter datatype [, parameter datatype])]

BEGIN

declaration_section

executable_section

END;

All variables which needs to be used must be declared at the start of procedure.

Executing Stored Procedure:

mariadb> call procedure_name;

Example:

```
delimiter //
CREATE PROCEDURE mypr1
BEGIN
  select "Hello World";
END;
//
delimiter ;
```

mariadb> call mypr1;

Declaring Variables:

Variables can be declared using declare statement.

Syntax:

declare <varname> <datatype> [DEFAULT <val>];

Ex:
declare x int;

Assigning Value to Variables:

set can be used to assign values to variables.

Syntax:

set <var> = <value>;

Example :

```
DELIMITER //

CREATE procedure mypr2()

BEGIN

  DECLARE total_value INT;

  SET total_value = 100;

  select total_value;
END; //

DELIMITER ;
```

Reading Values from Table and Storing in Variables

SELECT .. INTO <var> can be used to read values from tables and assign them to variables.

Syntax:

SELECT col1 INTO <var1> from .....

Example:

```
Select  count(*) INTO  cnt from student;

DELIMITER //

CREATE procedure mypr3()

BEGIN

  DECLARE cnt  INT;

  Select  count(*) INTO  cnt from student;


  select cnt as "Count";
END; //

DELIMITER ;
```

Example 2:

```
DELIMITER //

CREATE procedure mypr2()

BEGIN

  DECLARE x INT;
  DECLARE y varchar(50);
  select sid, sname into x, y from student where sid=10;
  select x,y;

END; //

DELIMITER ;
```

Passing Parameters to Procedure

MariaDB supports passing parameters to procedures. We can pass values and also procedure can return values using IN / OUT parameters.

Syntax:

CREATE procedure procedure_name(IN <paramater name> <data type>, ... OUT <parameter name> <data type> ...)

Example :

```
DELIMITER //

CREATE procedure mypr(IN sn int)

BEGIN
   DECLARE x INT;
  DECLARE y varchar(50);
    select sid, sname into x, y from student where sid=sn;
  select x,y;

END; //

DELIMITER ;
```

Example2:

```
DELIMITER //

CREATE procedure mypr(IN ci int, OUT cnt int)

BEGIN

      select count(*) into cnt  from student where courseid=ci;

END; //

DELIMITER ;
```

Execute the procedure with OUT parameter like below.

```
mariadb> call mypr(20, @a)
mariadb> select @a
```

Decision Making

Stored procedures supports IF conditions and CASE for decision making

Syntax of IF:

```
IF condition1 THEN
  Statements;

[ ELSEIF condition2 THEN
   statements; ]

[ ELSE
   statements
 ]

END IF;
```

Example:

```
DELIMITER //

CREATE procedure mypr4()

BEGIN

   DECLARE cnt  INT;

   Select  count(*) INTO  cnt from student;
   if  cnt  <  10 then
      select "Too Few rows";
   ELSEIF  cnt >= 10 and cnt < 50  then
       select "Average number of rows";
   ELSE
       select "Good number of rows";
  END IF;

END; //

DELIMITER ;
```

Loops

While Loop

While loop repeats the given code as long as the given condition remains true.

Syntax:

```
WHILE  condition  DO

  ...statements...

END WHILE ;
```

Example:

Calculate factorial of given number

```
DELIMITER //
CREATE PROCEDURE fa(IN x INT)
BEGIN

  DECLARE result INT;
  DECLARE i INT;
  SET result = 1;
  SET i = 1;

  WHILE i <= x DO
    SET result = result * i;
    SET i = i + 1;
  END WHILE;

  SELECT x AS Number, result as Factorial;

END;
//

DELIMITER ;
```

Until Loop

Until loop repeats the given code as long as the given condition remains false.

Syntax:

```
REPEAT

  ...statements...

UNTIL  <condition>
END REPEAT ;
```

Example:

Lets do the same factorial using Until Loop

```
DELIMITER //
CREATE or replace  PROCEDURE fact(IN x INT)
BEGIN

  DECLARE result INT;
  DECLARE i INT;
  SET result = 1;
```

```
  SET i = 1;
  REPEAT
    SET result = result * i;
    SET i = i + 1;
  UNTIL i > x
  END REPEAT;

  SELECT x AS Number, result as Factorial;

END;
//
DELIMITER ;
```

Cursors

MariaDB provides cursor for fetching table data into cursor and then allowing it to read row by row within the stored procedure.

Things to do to use cursors.

1. Declare a cursor
2. Declare a handler for cursor. Cursor handler is a special declaration which tracks the cursor reads and sets some variable when it reaches the last row.
3. Open cursor before reading data from cursor
4. Read data from cursor using FETCH
5. Once all rows from cursor are read CLOSE cursor.

Syntax for Declaring Cursor:

```
DECLARE  <cursorname>  CURSOR FOR  <select query>
```

Syntax for Declaring Handler for Cursor:

```
DECLARE CONTINUE HANDLER  FOR NOT FOUND SET <var> =  <value>;
```

Syntax for Opening Cursor:

```
OPEN  <cursor Name>;
```

Syntax for Fetching Data from Cursor:

```
FETCH  <cursor Name> INTO  <variable List>;
```

Syntax for Closing Cursor:

```
CLOSE <cursor Name>;
```

Cursor Example:

```
DELIMITER //

CREATE procedure my_cur()

BEGIN
   DECLARE x INT;
   DECLARE i INT DEFAULT 0;
```

```
  DECLARE y varchar(50);
  DECLARE mycr  CURSOR FOR  select sid, sname from student;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET  i = 1;
  OPEN mycr;
  while i = 0 do
   FETCH  mycr INTO x,y;
  select x,y;
  end while;
 CLOSE mycr;

END; //

DELIMITER ;
```

MariaDB Functions

MariaDb functions are like stored procedure except that they can return the value like system functions.

Syntax:

```
CREATE  FUNCTION function_name (parameter datatype [, parameter datatype])
RETURNS return_datatype
BEGIN
  declaration_section
  executable_section
   RETURN  <value>;
END;
```

Example:

```
DELIMITER //

CREATE function myfn()
returns int

BEGIN
     declare cnt int;

      select count(*) into cnt  from student;
      return cnt;

END; //
```

MariaDB Triggers

A trigger, as its name suggests, is a set of statements that run, or are triggered, when an event occurs on a table.

The event can be an INSERT, and UPDATE or a DELETE. The trigger can be executed BEFORE or AFTER the event.

Syntax:

```
CREATE [OR REPLACE]
     TRIGGER  trigger_name     BEFORE/AFTER       INSERT/UPDATE/DELETE
     ON tbl_name
     FOR EACH ROW
    BEGIN

        trigger_stmt

     END;
```

Example:

Lets write a DELETE trigger which writes the row into student2 table before deleting it.

```
DELIMITER //

CREATE TRIGGER  mytrg  BEFORE  DELETE ON student FOR EACH ROW
BEGIN
       insert into student2  (sid, sname, courseid) values (old.sid, old.sname, old.courseid);
END;
//
DELIMITER ;
```

Table Optimization & Statistics

Analyze Table
ANALYZE TABLE analyzes and stores the key distribution for a table. During the analysis, the table is locked with a read lock for InnoDB and MyISAM. This statement works with InnoDB, NDB, and MyISAM tables.

```
ANALYZE TABLE <table name>
```

Optimize Table
Reorganizes the physical storage of table data and associated index data, to reduce storage space and improve I/O efficiency when accessing the table.

```
OPTIMIZE TABLE <table name>
```

Query Optimization (Explain)

Explain
The EXPLAIN statement provides a way to obtain information about how MySQL executes a statement. It produces query execution plan as devised by the Optimizer. Explain does not fetches any data from tase nor it will modify any data.

```
EXPLAIN SELECT * FROM emp WHERE emp_no=1001\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

Columns EXPLAIN output

```
id            : Number identifying the examined
statement
select_type   : Type of select used in the query
SIMPLE, UNION or subqueries etc.
table         : Table for the output row
partitions    : Partitions that the optimizer
needs to examine to execute the query
type          : Access Method
possible_keys: Indexes that are relevant to the
query
key           : Specific index chosen by the
optimizer
key_len       : Size (in bytes) of all columns
in the index
ref           : Columns (or const) compared to
the index
rows          : Estimated number of rows that
the optimizer predicts will be returned by the
query
filtered      : Percentage of rows that are
filtered by the table condition
Extra         : Additional per-query information
provided by the optimizer or storage engine
```

Columns to watch from EXPLAIN output and Expected Values

```
select_type   : SIMPLE, UNION or subqueries etc.
table         : Table for the output row
partitions    : Partitions names
type          :
system: The SELECT statement is requesting data
from an in-memory table that contains a single
row.
const: Contains at most one matching row
eq_ref: Matches a single row
ref: Matches one or more rows
index: Index Scan
ALL: Full table Scan
possible_keys: Names of relevant indexes
key           : Selected Index name
Extra         :  Indicates any extra information
related query execution
```

Restructuring Queries to Improve Performance

- Rewriting inefficient SQL is often easier than repairing it
- Avoid expressions as the optimizer may ignore the indexes on such columns
- Use equi joins/ Inner join wherever possible to improve SQL efficiency
- Try changing the access path and join orders with hints
- Avoid full table scans if using an index is more efficient
- The join order can have a significant effect on performance

Normalization

Database Anomalies

Anomalies in Database design are faults in organizing data which may lead to in efficient way of managing database.

Update anomalies − If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
Deletion anomalies − We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
Insert anomalies − We tried to insert data in a record that does not exist at all.

Normalization

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

To achieve consistent database the below Normalization forms available

Un-Normalized Form (UNF)

If a table contains non-atomic values at each row, it is said to be in UNF. An atomic value is something that can not be further decomposed

Name	Age	Pet	Pet Name
Heather	10	dog, Cat	Rex, Thomas
Racheal	10	Cat	Fluff
Jimmy	11	Dog	Kimba
Lola	10	Cat	Thomas

First Normal Form (1NF)

A relation is said to be in 1NF if every attribute contains no non-atomic values and each row can provide a unique combination of values.

Name	Age	Pet	Pet Name
Heather	10	dog	Rex
Heather	10	Cat	Thomas
Racheal	10	Cat	Fluff

Jimmy	11	Dog	Kimba
Lola	10	Cat	Thomas

Second Normal Form (2NF)

A relation is said to be in 2NF if it is already in 1NF and each and every attribute fully depends on the primary key of the relation.

Name	Age
Heather	10
Racheal	10
Jimmy	11
Lola	10

Pet	Pet Name	Owner
dog	Rex	Heather
Cat	Thomas	Heather
Cat	Fluff	Racheal
Dog	Kimba	Jimmy
Cat	Thomas	Lola

Third Normal Form (3NF)

A relation is said to be in 3NF, if it is already in 2NF and there exists no transitive dependency in that relation.

ID	Name	Age
00	Heather	10
01	Racheal	10

02	Jimmy	11
03	Lola	10

Id	Pet	Pet Name	Owner ID
00	dog	Rex	00
01	Cat	Thomas	00
02	Cat	Fluff	01
03	Dog	Kimba	02
04	Cat	Thomas	03

Fourth Normal Form (4NF)

When attributes in a relation have multi-valued dependency, further Normalization to 4NF

ID	Name	Age
00	Heather	10
01	Rachael	10
02	Jimmy	11
03	Lola	10

Id	Pet	Pet Name
00	dog	Rex
01	Cat	Thomas
02	Cat	Fluff
03	Dog	Kimba

Pet-ID	Owner-ID
00	00
01	00
02	01
03	02
01	03