MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Agenda / Syllabus

- DBMS vs RDBMS
- MySQL: Introduction, Installation, …
- SQL
  - CREATE TABLE, MySQL data types
  - SELECT with LIMIT, ORDER, WHERE, GROUP BY, HAVING
  - INSERT, UPDATE, DELETE
  - Joins, Sub-queries
  - Transaction & Locking
  - GRANT & REVOKE
- MySQL programming (PSM) / PL-SQL
  - Stored procedure
  - Cursors
  - Functions
  - Triggers

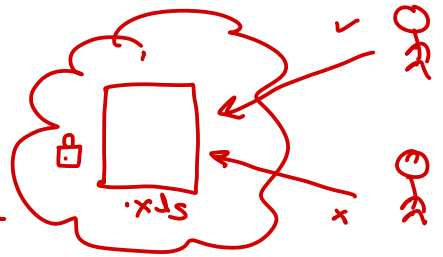
Syllabus : ① RDBMS : MySQL
② NoSQL : Mongo

Evaluation = 100 marks
Theory = 40 → CCEE (course end) MCQ
Lab = 40 → proctored lab exam, Lab assignments (module end)
Internals = 20 → MCQ

Interview Preparations
Interview Questions
- Rapid Fire
- Hot Seat

DBMS Database Management System.

- Any enterprise application need to manage data.
- In early days of software development, programmers store data into files and does operation on it. However data is highly application specific.
- Even today many software manage their data in custom formats e.g. Tally, Address book, etc.
- As data management became more common, DBMS systems were developed to handle the data. This enabled developers to focus on the business logic e.g. FoxPro, DBase, Excel, etc.
- At least CRUD (Create, Retrieve, Update and Delete) operations are supported by all databases.
- Traditional databases are file based, less secure, single-user, non-distributed, manage less amount of data (MB), complicated relation management, file-locking and need number of lines of code to use in applications.

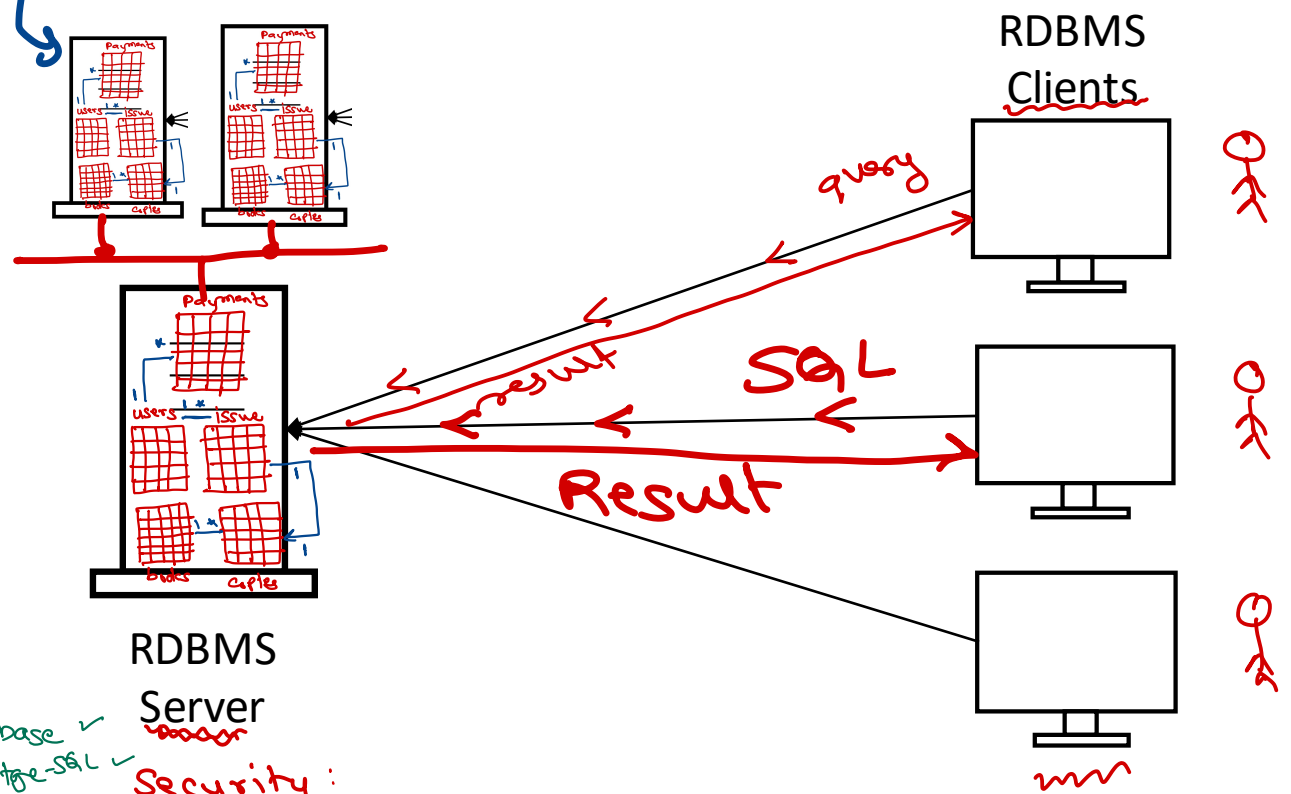


RDBMS

CRUD

DB cluster — Number of computers connected in a network for dedicated task.

- RDBMS is relational DBMS.
- It organizes data into Tables, rows and columns. The tables are related to each other.
- RDBMS follow table structure, more secure, multi-user, server-client architecture, server side processing, clustering support, manage huge data (TB), built-in relational capabilities, table-locking or row-locking and can be easily integrated with applications.
- e.g. DB2, Oracle, MS-SQL, MySQL, MS-Access, SQLite, … (mainframe, enterprise apps, open source; file based RDBMS (small data))
- RDBMS design is based on Codd's rules developed at IBM (in 1970). → Mathematician = 1 + 12 rules → based on set theory

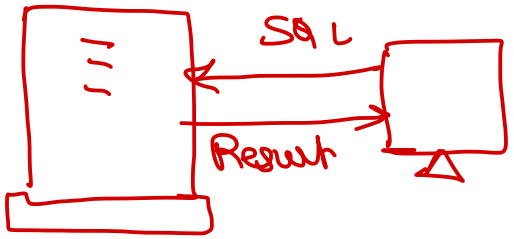


Cybase, Postgre-SQL, Informix, ...

Security:
OS + DB authentication
+ object level privileges
+ database/table/OPS.

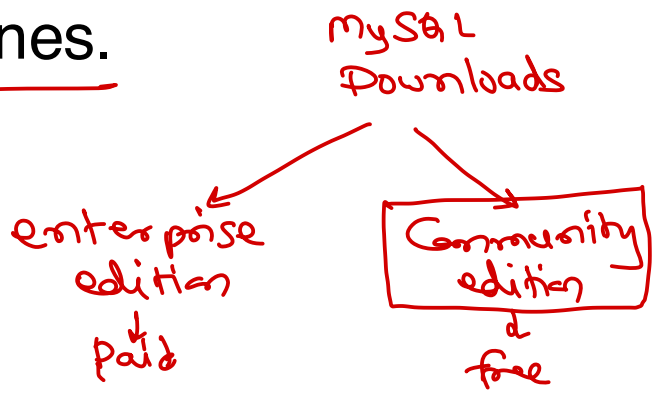
SQL : Structured Query Language

- Clients send SQL queries to RDBMS server and operations are performed accordingly.
- Originally it was named as RQBE (Relational Query By Example).
- SQL is ANSI standardised in 1987 and then revised multiple times adding new features. Recent revision in 2016.
- SQL is case insensitive.
- There are five major categories:
  - DDL: Data Definition Language e.g. CREATE, ALTER, DROP, RENAME.
  - DML: Data Manipulation Language e.g. INSERT, UPDATE, DELETE.
  - DQL: Data Query Language e.g. SELECT.
  - DCL: Data Control Language e.g. CREATE USER, GRANT, REVOKE.
  - TCL: Transaction Control Language e.g. SAVEPOINT, COMMIT, ROLLBACK.
- Table & column names allows alphabets, digits & few special symbols.
- If name contains special symbols then it should be back-quotes.
- e.g. Tbl1, `T1#`, `T2$` etc. Names can be max 30 chars long.



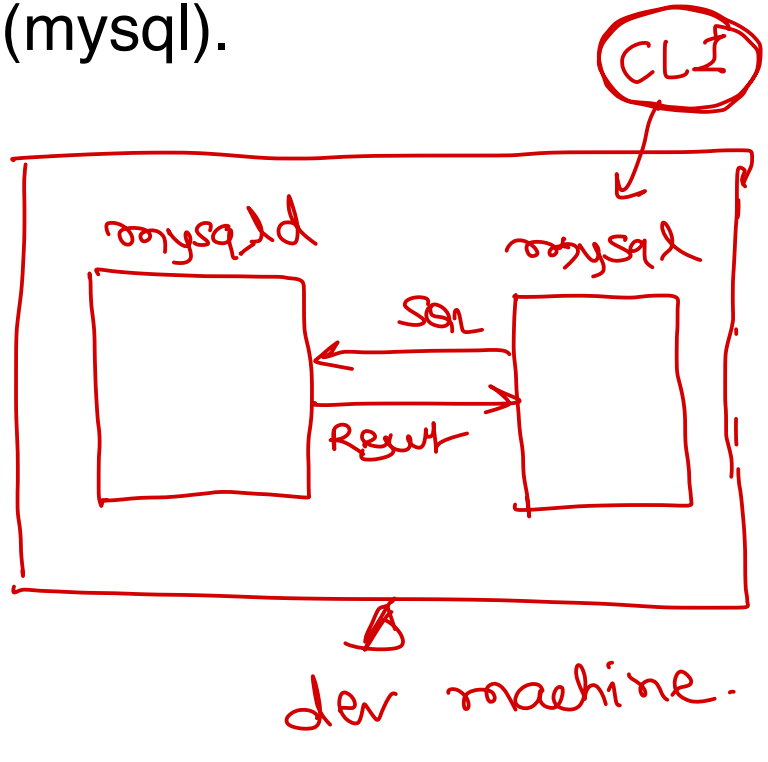
MySQL

- Developed by Michael Widenius in 1995. It is named after his daughter name Myia.
- Sun Microsystems acquired MySQL in 2008.
- Oracle acquired Sun Microsystem in 2010.
- MySQL is free and open-source database under GPL. However some enterprise modules are close sourced and available only under commercial version of MySQL.
- MariaDB is completely open-source clone of MySQL.
- MySQL support multiple database storage and processing engines.
- MySQL versions:
  - < 5.5: MyISAM storage engine
  - 5.5: InnoDb storage engine
  - 5.6: SQL Query optimizer improved, memcached style NoSQL
  - 5.7: Windowing functions, JSON data type added for flexible schema
  - 8.0: CTE, NoSQL document store.
- MySQL is database of year 2019 (in database engine ranking).



MySQL installation on Ubuntu/Linux

- terminal> sudo apt-get install mysql-community-server mysql-community-client
- This installs MySQL server (mysqld) and MySQL client (mysql).
- MySQL Server (mysqld)
  - Run as background process.
  - Implemented in C/C++.
  - Process SQL queries and generate results.
  - By default run on port 3306.
  - Controlled via systemctl.
    - terminal> sudo systemctl start|stop|status|enable|disable mysql
- MySQL client (mysql)
  - Command line interface
  - Send SQL queries to server and display its results.
  - terminal> mysql –u root -p
- Additional MySQL clients
  - MySQL workbench
  - PHPMyAdmin



Getting started

- root login can be used to perform CRUD as well as admin operations.
- It is recommended to create users for performing non-admin tasks.
  - mysql> CREATE DATABASE db;
  - mysql> SHOW DATABASES;
  - mysql> CREATE USER dbuser@localhost IDENTIFIED BY 'dbpass';
  - mysql> SELECT user, host FROM mysql.user;
  - mysql> GRANT ALL PRIVILEGES ON db.* TO dbuser@localhost;
  - mysql> FLUSH PRIVILEGES;
  - mysql> EXIT;
- terminal> mysql –u dbuser –pdbpass
  - mysql> SHOW DATABASES;
  - mysql> SELECT USER(), DATABASE();
  - mysql> USE db;
  - mysql> SHOW TABLES;
  - mysql> CREATE TABLE student(id INT, name VARCHAR(20), marks DOUBLE);
  - mysql> INSERT INTO student VALUES(1, 'Abc', 89.5);
  - mysql> SELECT * FROM student;

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

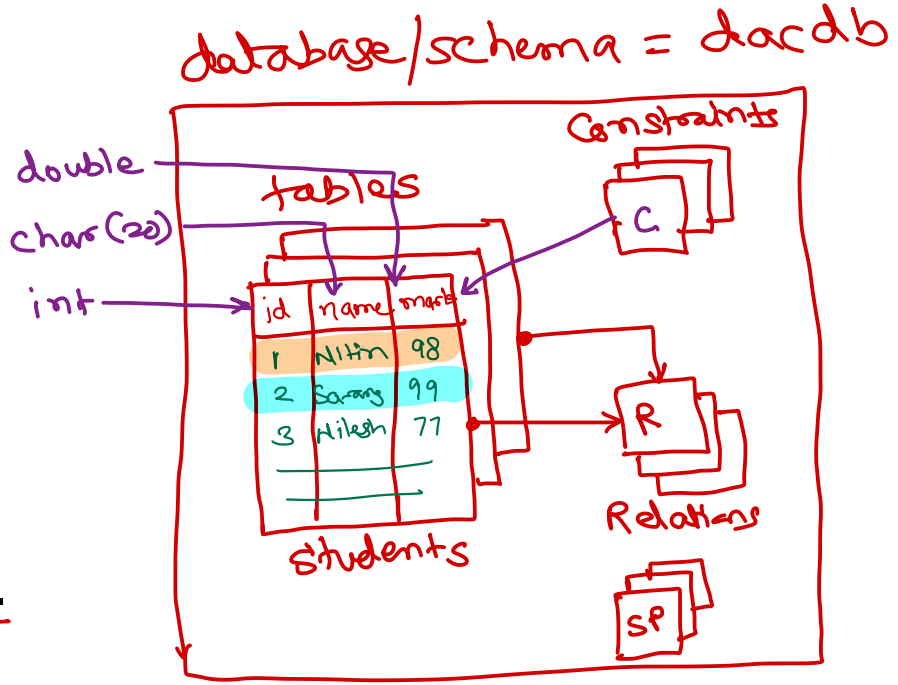
Getting started

- root login can be used to perform CRUD as well as admin operations.
- It is recommended to create users for performing non-admin tasks.
  - mysql> CREATE DATABASE db;
  - mysql> SHOW DATABASES;
  - mysql> CREATE USER dbuser@localhost IDENTIFIED BY 'dbpass';
  - mysql> SELECT user, host FROM mysql.user;
  - mysql> GRANT ALL PRIVILEGES ON db.* TO dbuser@localhost;
  - mysql> FLUSH PRIVILEGES;
  - mysql> EXIT;
- terminal> mysql –u dbuser –pdbpass
  - mysql> SHOW DATABASES;
  - mysql> SELECT USER(), DATABASE();
  - mysql> USE db;
  - mysql> SHOW TABLES;
  - mysql> CREATE TABLE student(id INT, name VARCHAR(20), marks DOUBLE);
  - mysql> INSERT INTO student VALUES(1, 'Abc', 89.5);
  - mysql> SELECT * FROM student;

Database logical layout

DESCRIBE students; → Show table structure (metadata).

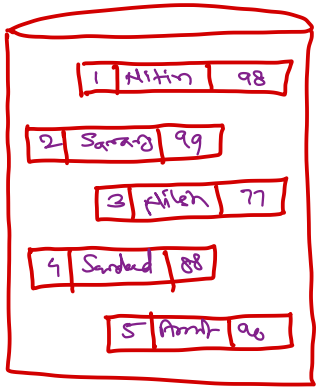
- Database/schema is like a namespace/container that stores all db objects related to a project.
- It contains tables, constraints, relations, stored procedures, functions, triggers, ...
- There are some system databases e.g. mysql, performance_schema, information_schema, sys, ... They contains db internal/system information.
  - e.g. SELECT user, host FROM mysql.user;
- A database contains one or more tables.
- Tables have multiple columns.
- Each column is associated with a data-type.
- Columns may have zero or more constraints.
- The data in table is in multiple rows.
- Each row have multiple values (as per columns).



Database physical layout

As a db developer, we must understand logical layout of the database. Physical layout understanding is only for info/GK.

- In MySQL, the data is stored on disk in its data directory i.e. /var/lib/mysql (Linux)
- Each database/schema is a separate sub-directory in data dir.
- Each table in the db, is a file on disk.
- e.g. student table in current db is stored in file /var/lib/mysql/db/student.ibd.
- Data is stored in binary format.
- A file may not be contiguously stored on hard disk.
- Data rows are not contiguous. They are scattered in the hard disk.
- In one row, all fields are consecutive.
- When records are selected, they are selected in any order.



SQL scripts

- SQL script is multiple SQL queries written into a .sql file.
- SQL scripts are mainly used while database backup and restore operations.
- SQL scripts can be executed from terminal as:
  - terminal> mysql –u user –ppassword db < /path/to/sqlfile
- SQL scripts can be executed from command line as:
  - mysql> SOURCE /path/to/sqlfile
- Note that SOURCE is MySQL CLI client command.
- It reads commands one by one from the script and execute them on server.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

MySQL data types

- RDBMS have similar data types (but not same).
- MySQL data types can be categorised as follows
  - Numeric types (Integers)
    - TINYINT (1 byte), SMALLINT (2 byte), MEDIUMINT (3 byte), INT (4 byte), BIGINT (8 byte), BIT(n bits)
    - integer types can signed (default) or unsigned.
  - Numeric types (Floating point)
    - approx. precision – FLOAT (4 byte), DOUBLE (8 byte) | DECIMAL(m, n) – exact precision
  - Date/Time types
    - DATE, TIME, DATETIME, TIMESTAMP, YEAR
  - String types – size = number of chars * size of char
    - CHAR(1-255) – Fixed length, Very fast access.
    - VARCHAR(1-65535) – Variable length, Stores length + chars.
    - TINYTEXT (255), TEXT (64K), MEDIUMTEXT (16M), LONGTEXT (4G) – Variable length, Slower access.
  - Binary types – size = number of bytes
    - BINARY, VARBINARY, TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
  - Miscellaneous types
    - ENUM, SET

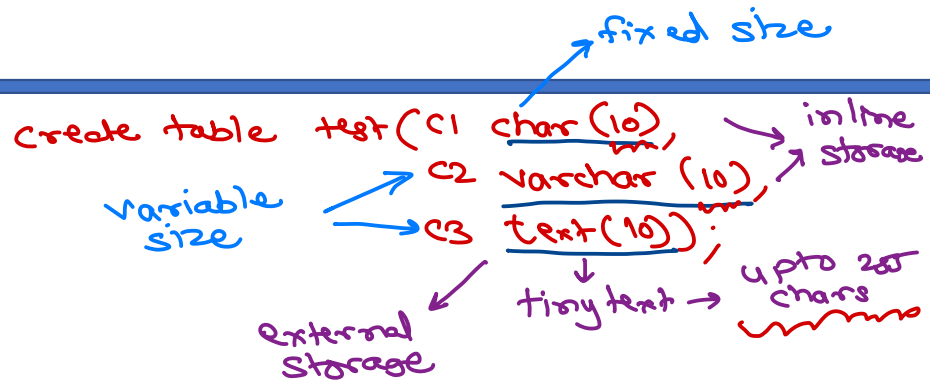
MySQL : INT, DECIMAL, FLOAT
Oracle : NUMBER
DERBY : INTEGER

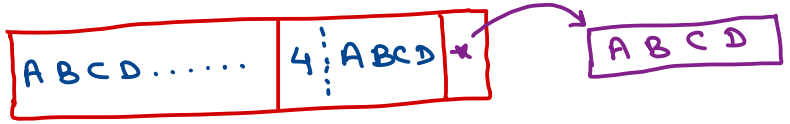
±2^15 ±2^23 ±2^31 ±2^63
±2^7
0 to 2^16 0 to 2^32
create table t1(c1 smallint unsigned, c2 int unsigned);
IEEE-754 format
create table t2(col decimal(8,3));
1 2 3 4 5.6 7 8
8

images, pdf, docs, ...
like text.

Size of char
depend on charset
1) ASCII → 1 byte
2) Unicode → 2 bytes
3) EBCDIC → 4 bytes (MBCS)

CHAR vs VARCHAR vs TEXT





- CHAR
  - Fixed inline storage.
  - If smaller data is given, rest of space is unused.
  - Very fast access.
- VARCHAR
  - Variable inline storage.
  - Stores length and characters.
  - Slower access than CHAR.
- TEXT
  - Variable external storage.
  - Very slow access.
  - Not ideal for indexing.
- CREATE TABLE temp(c1 CHAR(4), c2 VARCHAR(4), c3 TEXT(4));
- DESC temp;
- INSERT INTO temp VALUES('abcd', 'abcd', 'abcdef');

INSERT – DML

- Insert a new row (all columns, fixed order).
  - INSERT INTO table VALUES (v1, v2, v3);
- Insert a new row (specific columns, arbitrary order).
  - INSERT INTO table(c3, c1, c2) VALUES (v3, v1, v2);
  - INSERT INTO table(c1, c2) VALUES (v1, v2);
  - Missing columns data is NULL.
  - NULL is special value and it is not stored in database.
- Insert multiple rows.
  - INSERT INTO table VALUES (av1, av2, av3), (bv1, bv2, bv3), (cv1, cv2, cv3).
- Insert rows from another table.
  - INSERT INTO table SELECT c1, c2, c3 FROM another-table;
  - INSERT INTO table (c1,c2) SELECT c1, c2 FROM another-table;

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

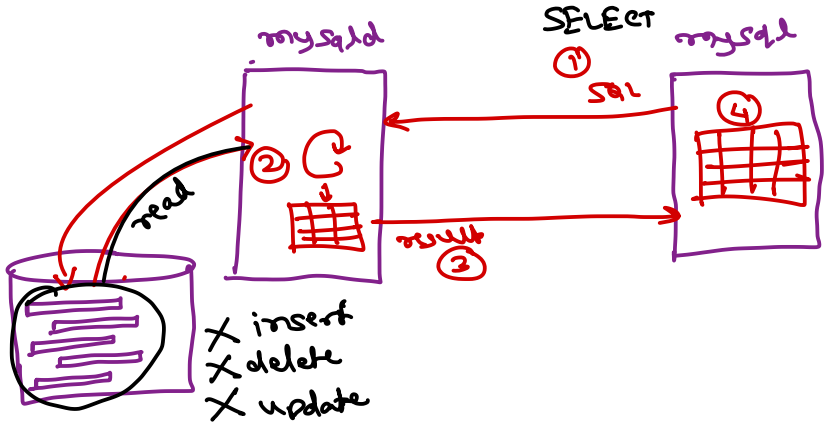
MySQL RDBMS

Trainer: Mr. Nilesh Ghule

SELECT – DQL

order in which cols added in table while creation.

- Select all columns (in fixed order).
  - SELECT * FROM table;
- Select specific columns / in arbitrary order.
  - SELECT c1, c2, c3 FROM table;
- Column alias
  - SELECT c1 AS col1, c2 col2 FROM table;
- Computed columns.
  - SELECT c1, c2, c3, expr1, expr2 FROM table;
  - SELECT c1,
  - CASE WHEN condition1 THEN value1,
  - CASE WHEN condition2 THEN value2,
  - …
  - ELSE valuen
  - END
  - FROM table;

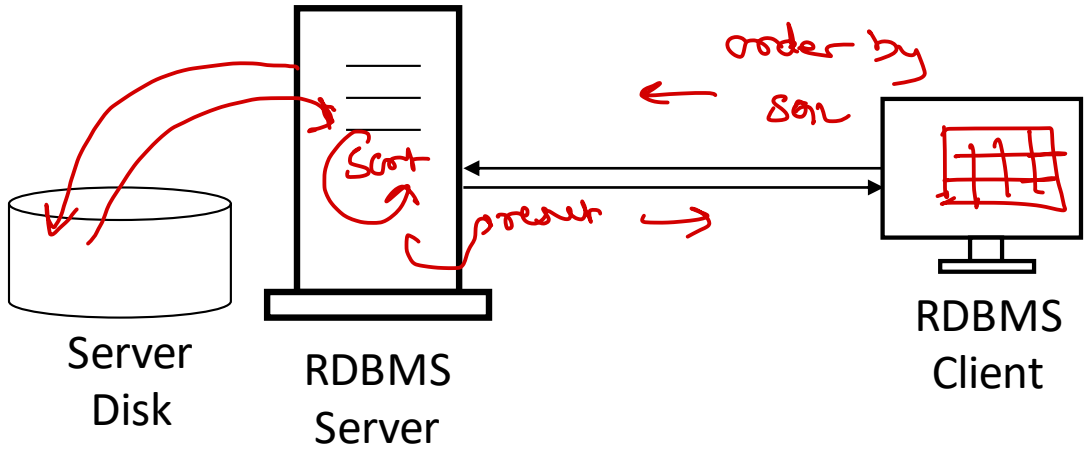


SELECT – DQL

- Distinct values in column.
  - SELECT DISTINCT c1 FROM table;
  - SELECT DISTINCT c1, c2 FROM table;
- Select limited rows.
  - SELECT * FROM table LIMIT n;
  - SELECT * FROM table LIMIT m, n;

SELECT – DQL – ORDER BY

- In db rows are scattered on disk. Hence may not be fetched in a fixed order.
- Select rows in asc order.
  - SELECT * FROM table ORDER BY c1;
  - SELECT * FROM table ORDER BY c2 ASC;
- Select rows in desc order.
  - SELECT * FROM table ORDER BY c3 DESC;
- Select rows sorted on multiple columns.
  - SELECT * FROM table ORDER BY c1, c2;
  - SELECT * FROM table ORDER BY c1 ASC, c2 DESC;
  - SELECT * FROM table ORDER BY c1 DESC, c2 DESC;
- Select top or bottom n rows.
  - SELECT * FROM table ORDER BY c1 ASC LIMIT n;
  - SELECT * FROM table ORDER BY c1 DESC LIMIT n;
  - SELECT * FROM table ORDER BY c1 ASC LIMIT m, n;



To sort too many records and/or sort on many columns will slow down execution of query.

SELECT – DQL – WHERE

- It is always good idea to fetch only required rows (to reduce network traffic).
- The WHERE clause is used to specify the condition, which records to be fetched.
- Relational operators
  - <, >, <=, >=, =, != or <>
- NULL related operators
  - NULL is special value and cannot be compared using relational operators.
  - IS NULL or <=>, IS NOT NULL.
- Logical operators
  - AND, OR, NOT

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

SELECT – DQL – WHERE

- It is always good idea to fetch only required rows (to reduce network traffic).
- The WHERE clause is used to specify the condition, which records to be fetched.
- Relational operators
  - <, >, <=, >=, =, != or <>
- NULL related operators
  - NULL is special value and cannot be compared using relational operators.
  - IS NULL or <=>, IS NOT NULL.
- Logical operators
  - AND, OR, NOT

SELECT – DQL – WHERE

- BETWEEN operator (include both ends)
  - c1 BETWEEN val1 AND val2
- IN operator (equality check with multiple values)
  - c1 IN (val1, val2, val3)
- LIKE operator (similar strings)
  - c1 LIKE ‘pattern’.
  - % represent any number of any characters.
  - _ represent any single character.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

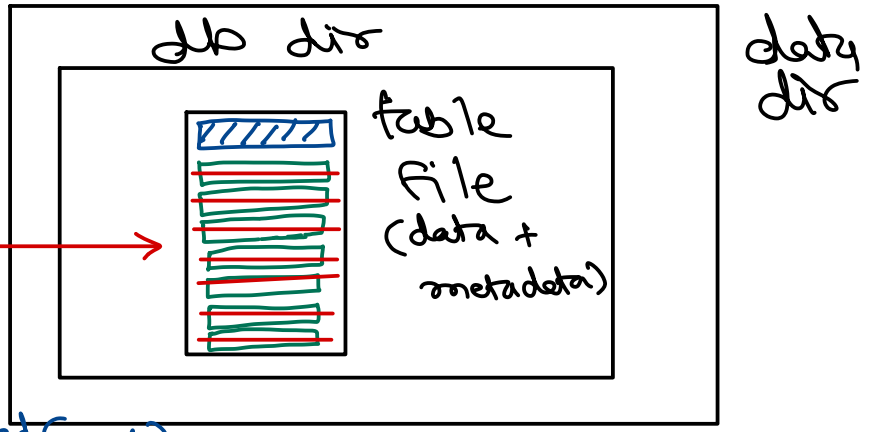
UPDATE – DML

- To change one or more rows in a table.
- Update row(s) single column.
  - UPDATE table SET c2=new-value WHERE c1=some-value;
- Update multiple columns.
  - UPDATE table SET c2=new-value, c3=new-value WHERE c1=some-value;
- Update all rows single column.
  - UPDATE table SET c2=new-value;

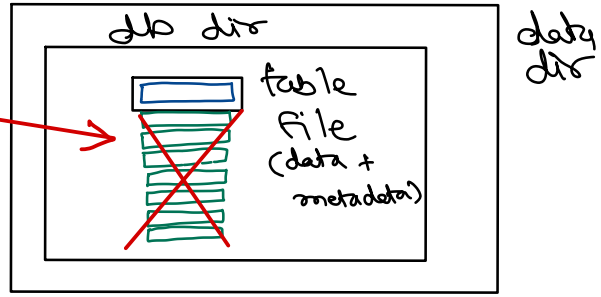
DELETE – DML vs TRUNCATE – DDL vs DROP – DDL

- DELETE
  - To delete one or more rows in a table.
  - Delete row(s)
    - DELETE FROM table WHERE c1=value;
  - Delete all rows
    - DELETE FROM table
- TRUNCATE
  - Delete all rows.
    - TRUNCATE TABLE table;
  - Truncate is faster than DELETE.
- DROP
  - Delete all rows as well as table structure.
    - DROP TABLE table;
    - DROP TABLE table IF EXISTS;
  - Delete database/schema.
    - DROP DATABASE db;

Query impl change from RDBMS to RDBMS.
In general - RDBMS.



① mark all rows as deleted.
② space occupied by them can be overwritten/reused by new records.
③ actual table file size is not changed (much).
④ DML query - roll backed.



① truncate file size so that only structure is kept.
② all rows space is released.
③ much faster operation for huge table.
④ DDL query - can never be rollbacked.

① delete table file struct + data.
② DDL - no rollback.

Synonym for DATABASE ← SCHEMA → dbname

Seeking HELP

C R U D

- HELP is client command to seek help on commands/functions.
  - HELP SELECT;
  - HELP Functions;
  - HELP SIGN;

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Seeking HELP

C R U D

- HELP is client command to seek help on commands/functions.
  - HELP SELECT;
  - HELP Functions;
  - HELP SIGN;

DUAL table → First used in oracle. Two rows table. Later it made single row & col table.

- A dummy/in-memory a table having single row & single column.
- It is used for arbitrary calculations, testing functions, etc.
  - SELECT 2 + 3 * 4 FROM DUAL;
  - SELECT NOW() FROM DUAL;
  - SELECT USER(), DATABASE() FROM DUAL;
- In MySQL, DUAL keyword is optional.
  - SELECT 2 + 3 * 4;
  - SELECT NOW();
  - SELECT USER(), DATABASE();

Numeric & String functions

- ABS()
- POWER()
- ROUND(), FLOOR(), CEIL()

- ASCII(), CHAR()
- CONCAT()
- SUBSTRING()
- LOWER(), UPPER()
- TRIM(), LTRIM(), RTRIM()
- LPAD(), RPAD()
- REGEXP_LIKE()

Thank you!

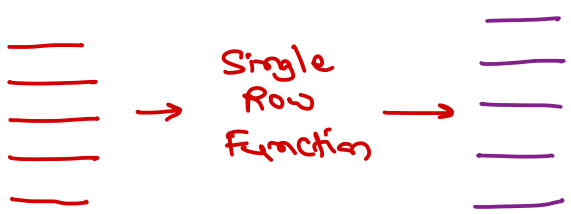
Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

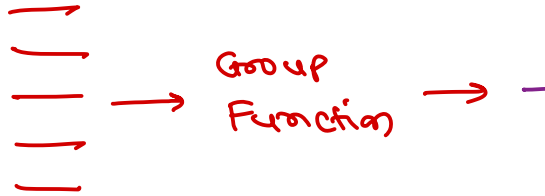
Trainer: Mr. Nilesh Ghule

SQL functions

- RDBMS provides many built-in functions to process the data.
- These functions can be classified as:
  - Single row functions
    - One row input produce one row output.
    - e.g. ABS(), CONCAT(), IFNULL(), …
  - Multi-row or Group functions or aggregate functions
    - Values from multiple rows are aggregated to single value.
    - e.g. SUM(), MIN(), MAX(), …



function work on one or more values/columns from same row of the table.



function work on multiple values from multiple rows

- These functions can also be categorized based on data types or usage.
  - Numeric functions
  - String functions
  - Date and Time functions
  - Control flow functions
  - Information functions
  - Miscellaneous functions

Numeric & String functions

- ABS()
- POWER()
- ROUND(), FLOOR(), CEIL()

- ASCII(), CHAR()
- CONCAT()
- SUBSTRING()
- LOWER(), UPPER()
- TRIM(), LTRIM(), RTRIM()
- LPAD(), RPAD()
- REGEXP_LIKE()

Date-Time and Information functions

- VERSION()
- USER(), DATABASE()

- MySQL supports multiple date time related data types
  - DATE (3), TIME (3), DATETIME (5), TIMESTAMP (4), YEAR (1)
- SYSDATE(), NOW()
- DATE(), TIME()
- DAYOFMONTH(), MONTH(), YEAR(), HOUR(), MINUTE(), SECOND(), …
- DATEDIFF(), DATE_ADD(), TIMEDIFF()
- MAKEDATE(), MAKETIME()

Control and NULL and List functions

- NULL is special value in RDBMS that represents absence of value in that column.
- NULL values do not work with relational operators and need to use special operators.
- Most of functions return NULL if NULL value is passed as one of its argument.
- ISNULL()
- IFNULL()
- NULLIF()
- COALESCE()

- GREATEST(), LEAST()

- IF(condition, true-value, false-value)

Group functions

- Work on group of rows of table.
- Input to function is data from multiple rows & then output is single row. Hence these functions are called as "Multi Row Function“ or "Group Functions“.
- These functions are used to perform aggregate ops like sum, avg, max, min, count or std dev, etc. Hence these fns are also called as "Aggregate Functions".
- Example: SUM(), AVG(), MAX(), MIN(), COUNT().
- NULL values are ignored by group functions.
- Limitations of GROUP functions:
  - Cannot select group function along with a column.
  - Cannot select group function along with a single row fn.
  - Cannot use group function in WHERE clause/condition.
  - Cannot nest a group function in another group fn.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

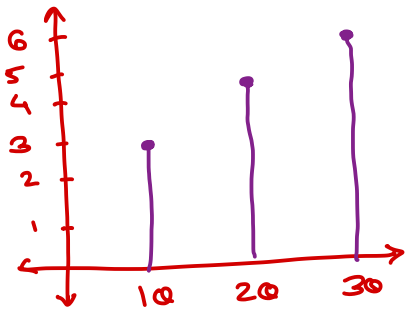
Trainer: Mr. Nilesh Ghule

Group functions

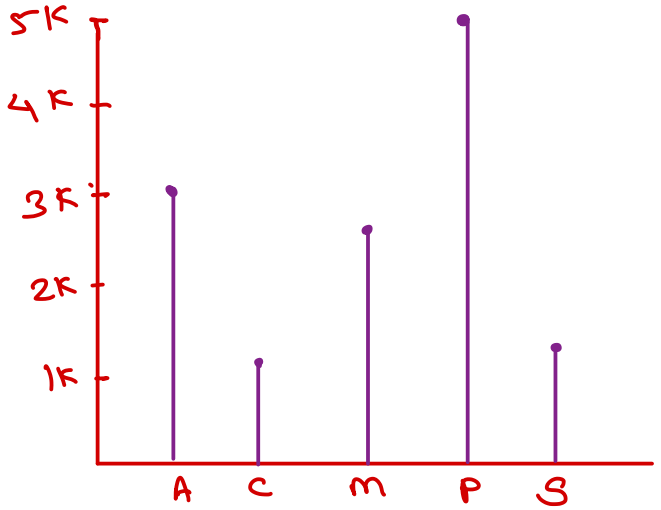
- Work on group of rows of table.
- Input to function is data from multiple rows & then output is single row. Hence these functions are called as "Multi Row Function“ or "Group Functions“.
- These functions are used to perform aggregate ops like sum, avg, max, min, count or std dev, etc. Hence these fns are also called as "Aggregate Functions".
- Example: SUM(), AVG(), MAX(), MIN(), COUNT().
- NULL values are ignored by group functions.
- Limitations of GROUP functions:
  - Cannot select group function along with a column.
  - Cannot select group function along with a single row fn.
  - Cannot use group function in WHERE clause/condition.
  - Cannot nest a group function in another group fn.

GROUP BY clause

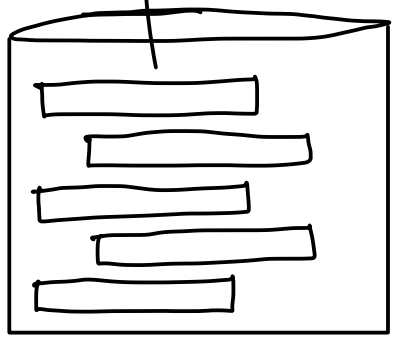
- GROUP BY is used for analysis of data i.e. generating reports & charts.
- When GROUP BY single column, generated output can be used to plot 2-D chart.
  When GROUP BY two column, generated output can be used to plot 3-D chart and so on.
- GROUP BY queries are also called as Multi-dimensional / Spatial queries.
- Syntactical Characteristics:
  - If a column is used for GROUP BY, then it may or may not be used in SELECT clause.
  - If a column is in SELECT, it must be in GROUP BY.
- When GROUP BY query is fired on database server, it does following:
  - Load data from server disk into server RAM.
  - Sort data on group by columns.
  - Group similar records by group columns.
  - Perform given aggregate ops on each column.
  - Send result to client.

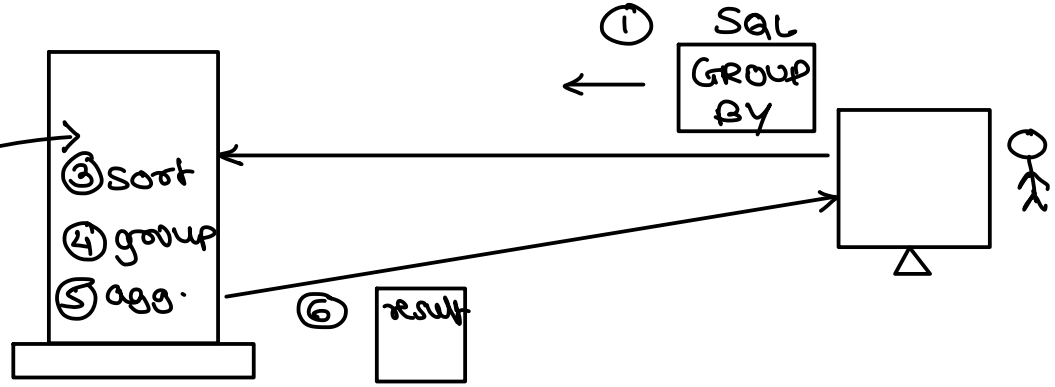



```
SELECT deptno, COUNT(empno)
FROM emp
GROUP BY deptno;
```




```
SELECT job, AVG(sal)
FROM emp
GROUP BY job;
```





Group by on multiple columns and/or too many rows slow down execution.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

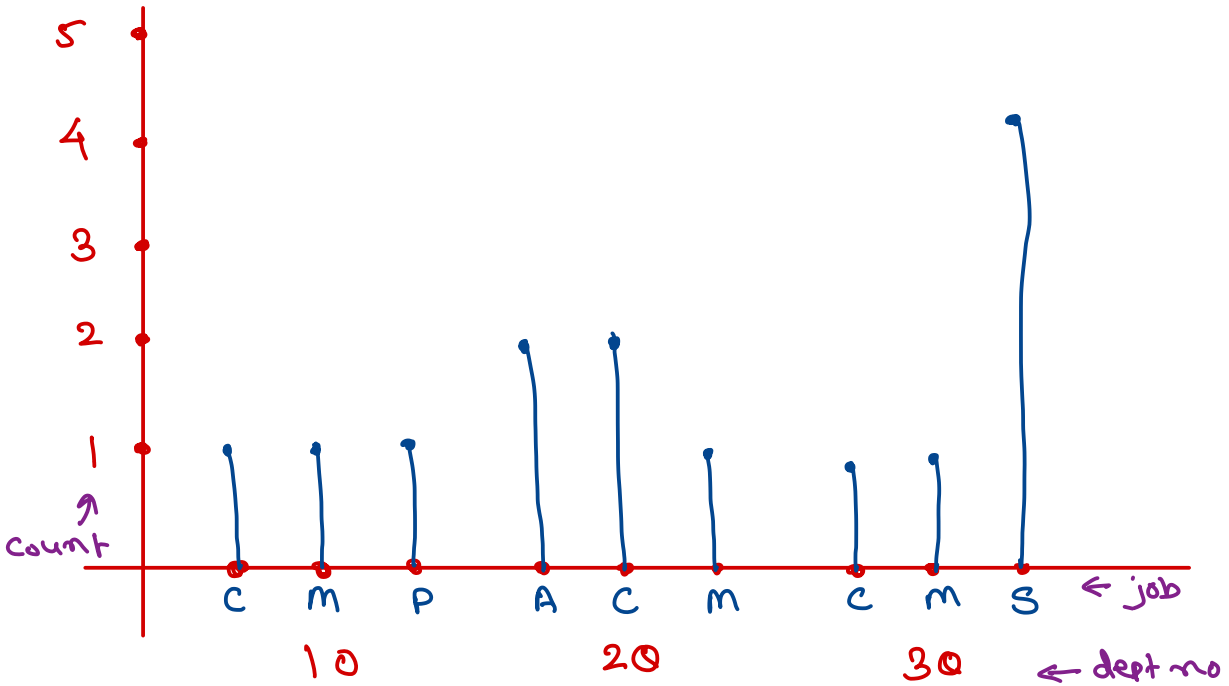
MySQL RDBMS

Trainer: Mr. Nilesh Ghule

GROUP BY clause

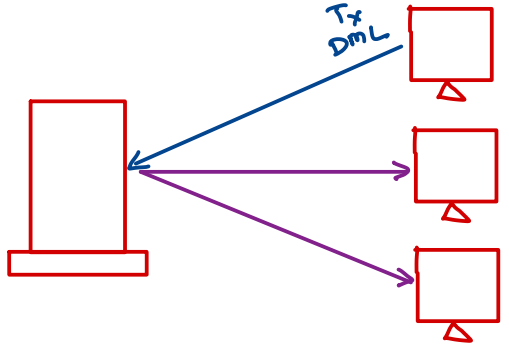
- GROUP BY is used for analysis of data i.e. generating reports & charts.
- When GROUP BY single column, generated output can be used to plot 2-D chart. When GROUP BY two column, generated output can be used to plot 3-D chart and so on.
- GROUP BY queries are also called as Multi-dimensional / Spatial queries.
- Syntactical Characteristics:
  - If a column is used for GROUP BY, then it may or may not be used in SELECT clause.
  - If a column is in SELECT, it must be in GROUP BY.
- When GROUP BY query is fired on database server, it does following:
  - Load data from server disk into server RAM.
  - Sort data on group by columns.
  - Group similar records by group columns.
  - Perform given aggregate ops on each column.
  - Send result to client.

```
select deptno, job, count(empno)
from emp
group by deptno, job;
```

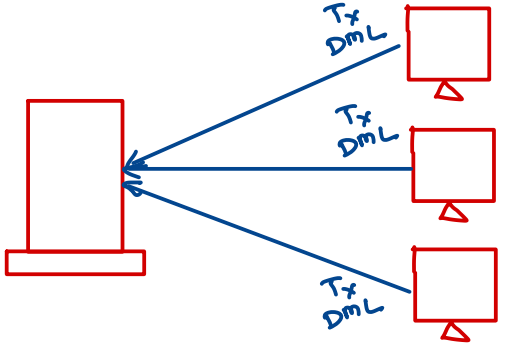


Transaction

- Transaction is set of DML queries executed as a single unit.
- Transaction examples
  - accounts table [id, type, balance]
  - UPDATE accounts SET balance=balance-1000 WHERE id = 1;
  - UPDATE accounts SET balance=balance+1000 WHERE id = 2;



- RDBMS transaction have ACID properties.
  - Atomicity
    - All queries are executed as a single unit. If any query is failed, other queries are discarded.
  - Consistency
    - When transaction is completed, all clients see the same data.
  - Isolation
    - Multiple transactions (by same or multiple clients) are processed concurrently.
  - Durable
    - When transaction is completed, all data is saved on disk.



Transaction

- Transaction management
  - START TRANSACTION;
  - … dml1 dml2 dml3
  - COMMIT WORK; → final save

  - START TRANSACTION;
  - … dml1; dml2; dml3;
  - ROLLBACK WORK; → discard
- In MySQL autocommit variable is by default 1. So each DML command is auto-committed into database.
  - SELECT @@autocommit;
- Changing autocommit to 0, will create new transaction immediately after current transaction is completed. This setting can be made permanent in config file.
  - SET autocommit=0;

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

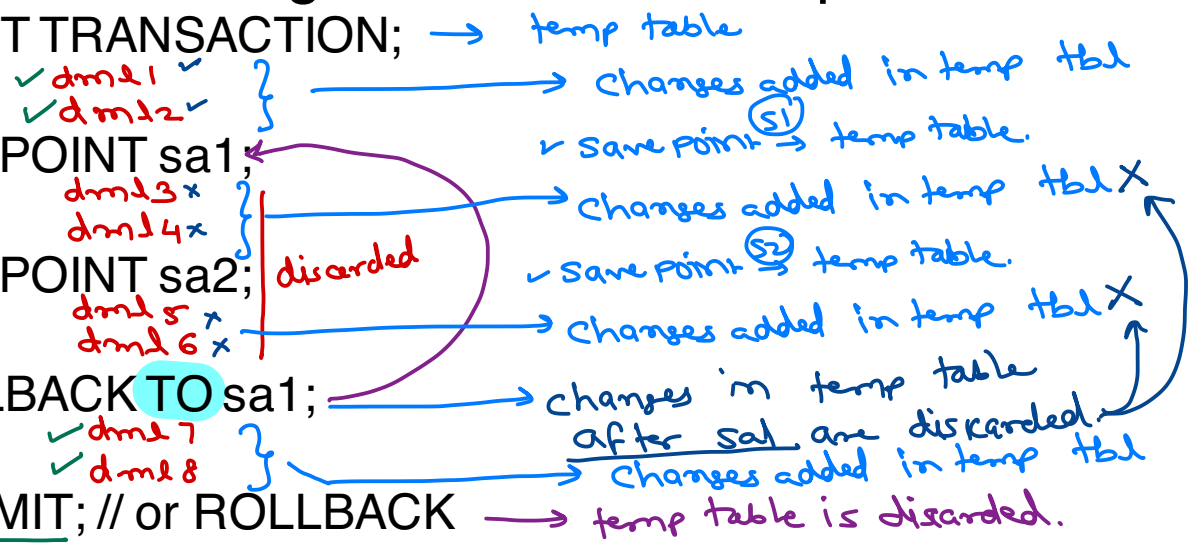
Transaction

- Transaction management
  - START TRANSACTION;
  - … dml 1 dml 2 dml 3
  - COMMIT WORK; → final save

  - START TRANSACTION;
  - … dml 1; dml 2; dml 3;
  - ROLLBACK WORK; → discard
- In MySQL autocommit variable is by default 1. So each DML command is auto-committed into database.
  - SELECT @@autocommit;
- Changing autocommit to 0, will create new transaction immediately after current transaction is completed. This setting can be made permanent in config file.
  - SET autocommit=0;

Transaction

- Save-point is state of database tables (data) at the moment (within a transaction).
- It is advised to create save-points at end of each logical section of work.
- Database user may choose to rollback to any of the save-point.
- Transaction management with Save-points
  - START TRANSACTION;
  - …
  - SAVEPOINT sa1;
  - …
  - SAVEPOINT sa2;
  - …
  - ROLLBACK TO sa1;
  - …
  - COMMIT; // or ROLLBACK



- Commit always commit the whole transaction.
- ROLLBACK or COMMIT clears all save-points.

Transaction

- Transaction is set of DML statements.
- If any DDL statement is executed, current transaction is automatically committed.
- Any power failure, system or network failure automatically rollback current state.
- Transactions are isolated from each other and are consistent.

Row locking

- When an user update or delete a row (within a transaction), that row is locked and becomes read-only for other users.
- The other users see old row values, until transaction is committed by first user.
- If other users try to modify or delete such locked row, their transaction processing is blocked until row is unlocked.
- Other users can INSERT into that table. Also they can UPDATE or DELETE other rows.
- The locks are automatically released when COMMIT/ROLLBACK is done by the user.
- This whole process is done automatically in MySQL. It is called as "OPTIMISTIC LOCKING".

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

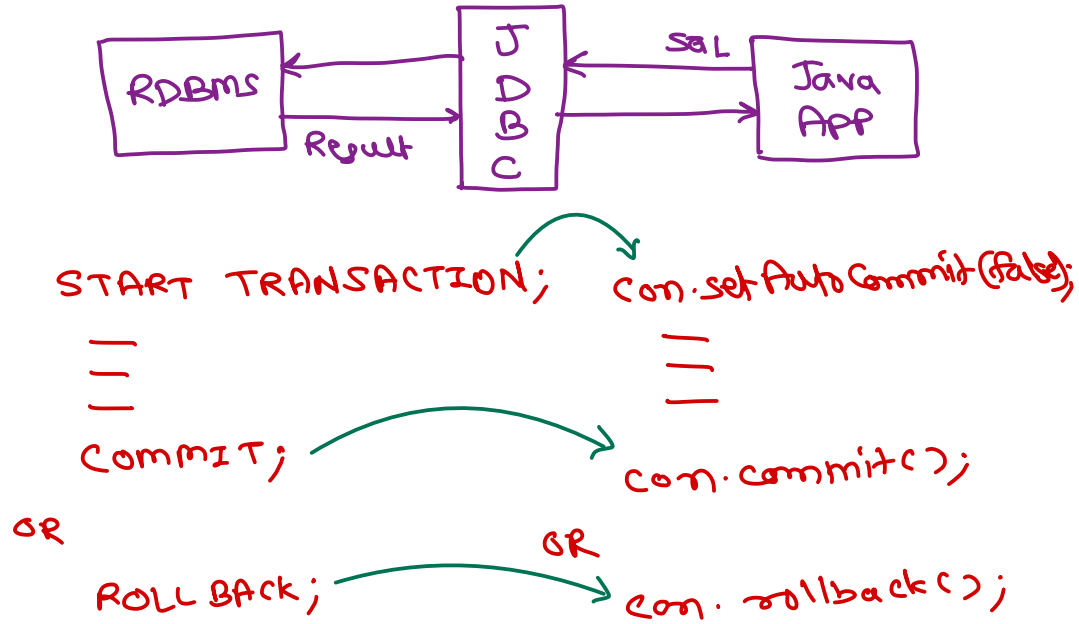
Trainer: Mr. Nilesh Ghule

Row locking

- When an user update or delete a row (within a transaction), that row is locked and becomes read-only for other users.
- The other users see old row values, until transaction is committed by first user.
- If other users try to modify or delete such locked row, their transaction processing is blocked until row is unlocked. → or timeout.
- Other users can INSERT into that table. Also they can UPDATE or DELETE other rows.
- The locks are automatically released when COMMIT/ROLLBACK is done by the user. → first.
- This whole process is done automatically in MySQL. It is called as "OPTIMISTIC LOCKING".

Row locking

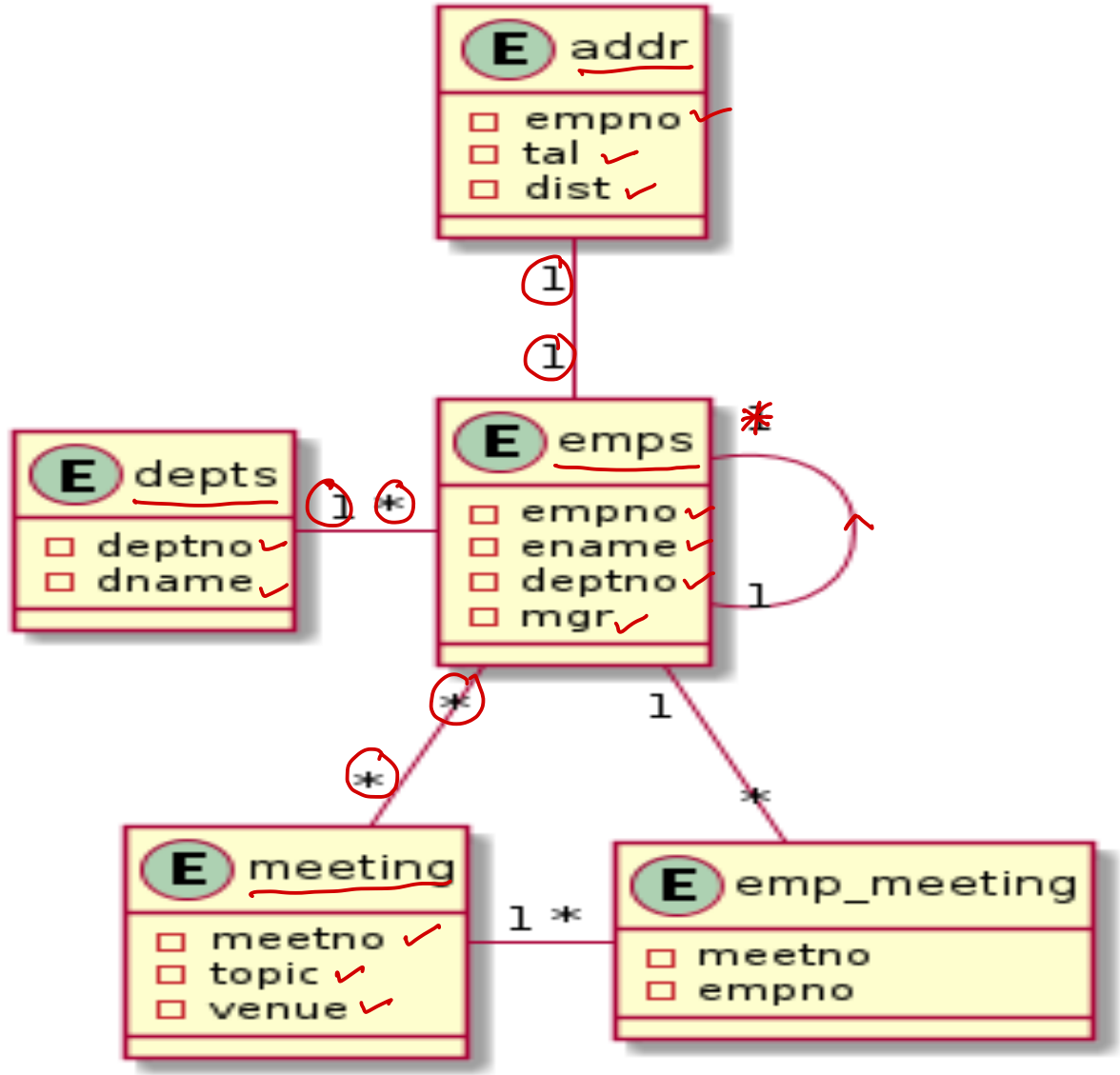
- Manually locking the row in advanced before issuing UPDATE or DELETE is known as "PESSIMISTIC LOCKING".
- This is done by appending FOR UPDATE to the SELECT query.
- It will lock all selected rows, until transaction is committed or rollbacked. (or timeout)
- If these rows are already locked by another users, the SELECT operation is blocked until rows lock is released.



- By default MySQL does table locking. Row locking is possible only when table is indexed on the column.

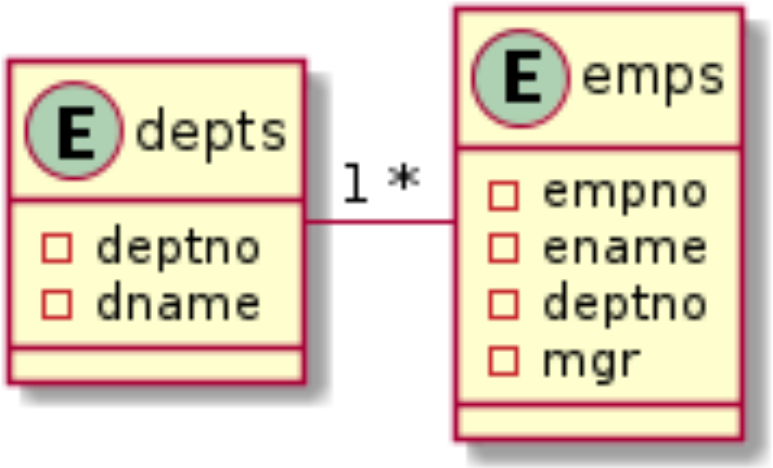
Entity Relations

- To avoid redundancy of the data, data should be organized into multiple tables so that tables are related to each other.
- The relations can be one of the following
  - One to One → emps → addr
  - One to Many → depts → emps
  - Many to One → emps → depts
  - Many to Many → emps ↔ meeting
- Entity relations is outcome of Normalization process.



SQL Joins

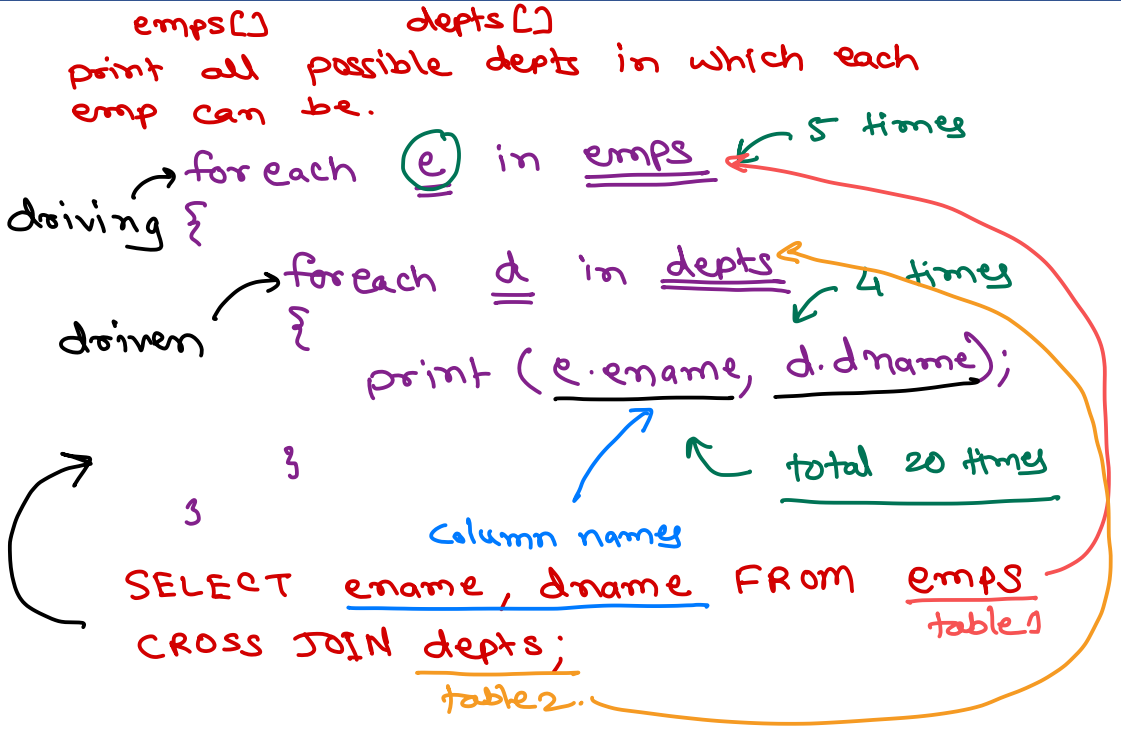
- Join statements are used to SELECT data from multiple tables using single query.
- Typical RDBMS supports following types of joins:
  - Cross Join
  - Inner Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join
  - Self join



Cross Join

deptno	dname
10	DEV
20	QA
30	OPS
40	ACC

empno	ename	deptno
1	Amit	10
2	Rahul	10
3	Nilesh	20
4	Nitin	50
5	Sarang	50

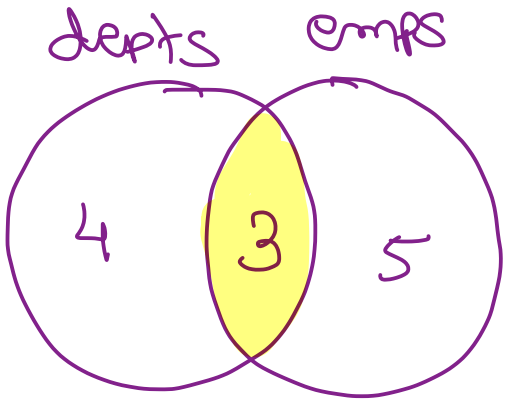


- Compares each row of Table1 with every row of Table2.
- Yields all possible combinations of Table1 and Table2.
- In MySQL, The larger table is referred as "Driving Table", while smaller table is referred as "Driven Table". Each row of Driving table is combined with every row of Driven table.
- Cross join is the fastest join, because there is no condition check involved.

Inner Join

deptno	dname
10	DEV
20	QA
30	OPS
40	ACC

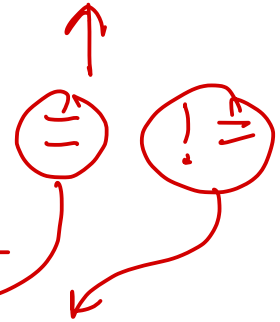
empno	ename	deptno
1	Amit	10
2	Rahul	10
3	Nilesh	20
4	Nitin	50
5	Sarang	50




```
foreach e in emps
{
    foreach d in depts
    {
        if (e.deptno == d.deptno)
            print (e.ename, d.dname);
    }
}
```

Columns

```
SELECT e.ename, d.dname FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno;
```



- The inner JOIN is used to return rows from both tables that satisfy the join condition.
- Non-matching rows from both tables are skipped.
- If join condition contains equality check, it is referred as equi-join; otherwise it is non-equi-join.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

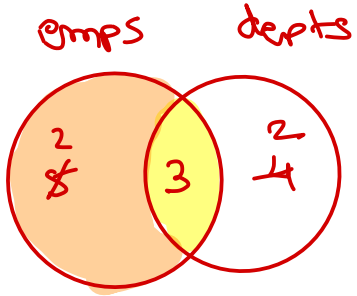
Trainer: Mr. Nilesh Ghule

Left Outer Join

deptno	dname
10	DEV
20	QA
30	OPS
40	ACC

empno	ename	deptno
1	Amit	10
2	Rahul	10
3	Nilesh	20
4	Nitin	50
5	Sarang	50

```
foreach e in emps
{
    found = false;
    foreach d in depts
    {
        if (e.deptno == d.deptno) {
            print (e.ename, d.dname);
            found = true;
        }
    }
    if ( found == false)
        print (e.ename, NULL);
}
```




```
Select e.ename, d.dname from emps e
left outer join depts d on e.deptno = d.deptno;
```

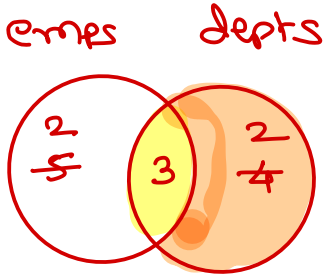
(left = emps e; right = depts d)

- Left outer join is used to return matching rows from both tables along with additional rows in left table.
- Corresponding to additional rows in left table, right table values are taken as NULL.
- OUTER keyword is optional.

Right Outer Join

deptno	dname
10	DEV
20	QA
30	OPS
40	ACC

empno	ename	deptno
1	Amit	10
2	Rahul	10
3	Nilesh	20
4	Nitin	50
5	Sarang	50




```
Select e.ename, d.dname from emps e
right outer join depts d on e.deptno=d.deptno;
```

left

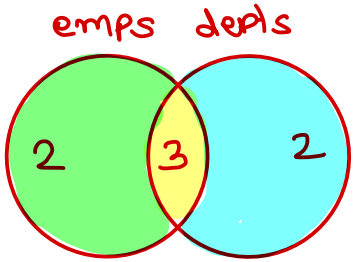
right

- Right outer join is used to return matching rows from both tables along with additional rows in right table.
- Corresponding to additional rows in right table, left table values are taken as NULL.
- OUTER keyword is optional.

Full Outer Join

deptno	dname
10	DEV
20	QA
30	OPS
40	ACC

empno	ename	deptno
1	Amit	10
2	Rahul	10
3	Nilesh	20
4	Nitin	50
5	Sarang	50

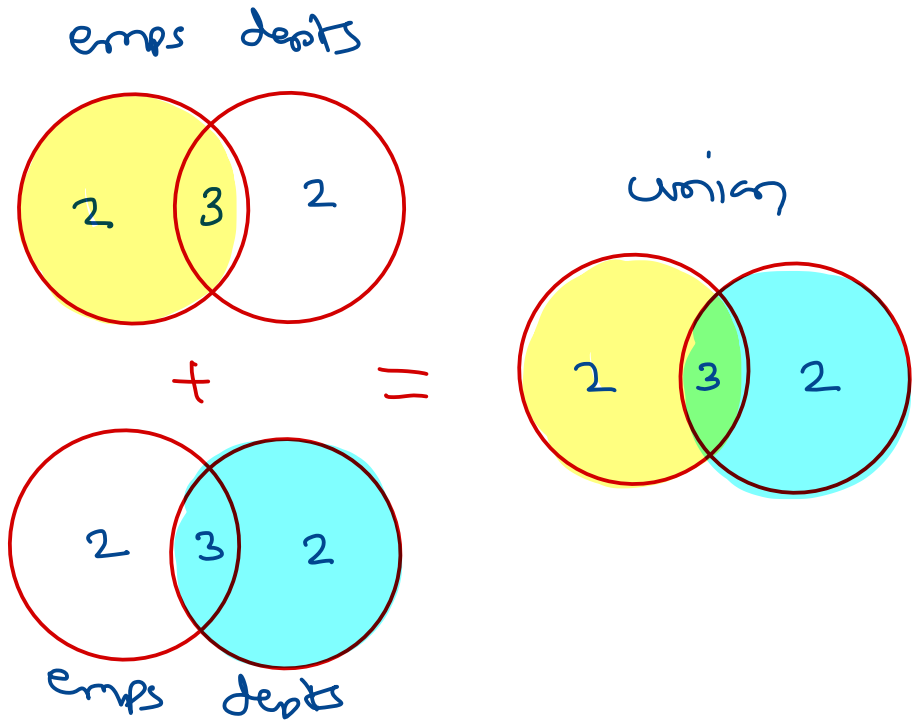


- Full join is used to return matching rows from both tables along with additional rows in both tables.
- Corresponding to additional rows in left or right table, opposite table values are taken as NULL.
- Full outer join is not supported in MySQL, but can be simulated using set operators.

Set operators

ename	dname
Amit	DEV
Rahul	DEV
Nilesh	QA
NULL	OPS
NULL	ACC

ename	dname
Amit	DEV
Rahul	DEV
Nilesh	QA
Nitin	NULL
Sarang	NULL



- UNION operator is used to combine results of two queries. The common data is taken only once. It can be used to simulate full outer join.
- UNION ALL operator is used to combine results of two queries. Common data is repeated.

Self Join

- When join is done on same table, then it is known as "Self Join". The both columns in condition belong to the same table.
- Self join may be an inner join or outer join.

empno	ename	deptno	mgr
1	Amit	10	4
2	Rahul	10	3
3	Nilesh	20	4
4	Nitin	50	5
5	Sarang	50	NULL

empno	ename	deptno	mgr
1	Amit	10	4
2	Rahul	10	3
3	Nilesh	20	4
4	Nitin	50	5
5	Sarang	50	NULL

emps e

Amit - Nitin
Rahul - Nilesh
Nilesh - Nitin
Nitin - Sarang

Sarang
|
Nitin
Nilesh Amit
|
Rahul

mgrs m

```
emps: | 1 | 2 | 3 | 4 | 5 |

foreach e in emps
{
    foreach m in emps
    {
        if(e.mgr == m.empno)
            print(e.ename, m.ename);
    }
}
```

```
select e.ename, m.ename from emps e
inner join emps m on e.mgr = m.empno;
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

DDL – ALTER statement ALTER, MODIFY, RENAME, DROP

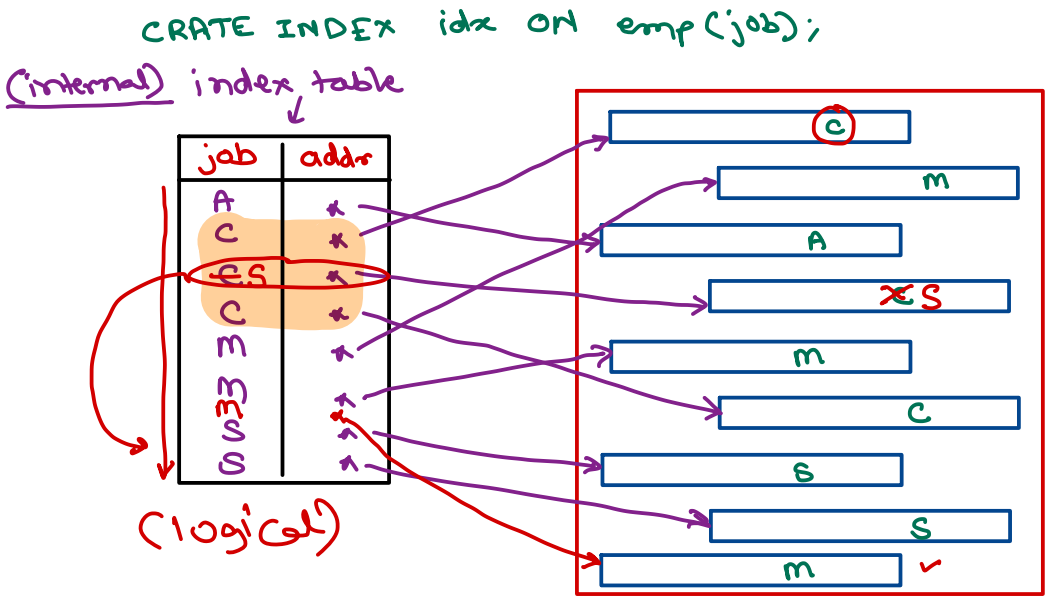
- ALTER statement is used to do modification into table, view, function, procedure, …
- ALTER TABLE is used to change table structure.
- Add new column(s) into the table.
  - ALTER TABLE table ADD col TYPE;
  - ALTER TABLE table ADD c1 TYPE, c2 TYPE;
- Modify column of the table.
  - ALTER TABLE table MODIFY col NEW_TYPE;
- Rename column.
  - ALTER TABLE CHANGE old_col new_col TYPE;
- Drop a column
  - ALTER TABLE DROP COLUMN col;
- Rename table
  - ALTER TABLE table RENAME TO new_table;

Using ALTER TABLE on production db is bad practice / strictly prohibited.

Index

- Index enable faster searching in tables by indexed columns.
  - CREATE INDEX idx_name ON table(column);
- One table can have multiple indexes on different columns/order.
- Typically indexes are stored as some data structure (like BTREE or HASH) on disk.
- Indexes are updated during DML operations. So DML operation are slower on indexed tables.

automatically



Query performance

- Few RDBMS features ensure better query performance.
  - Index speed up execution of SELECT queries (search operations).
  - Correlated sub-queries execute faster.
- Query performance can observed using EXPLAIN statement.
  - EXPLAIN FORMAT=JSON SELECT …;
- EXPLAIN statement shows
  - Query cost (Lower is the cost, faster is the query execution).
  - Execution plan (Algorithm used to execute query e.g. loop, semi-join, materialization, etc).
- Optimizations can be enabled or disabled by optimizer_switch system variable.
  - SELECT @@optimizer_switch;
  - SET @@optimizer_switch='materialization=off';

Thank you!

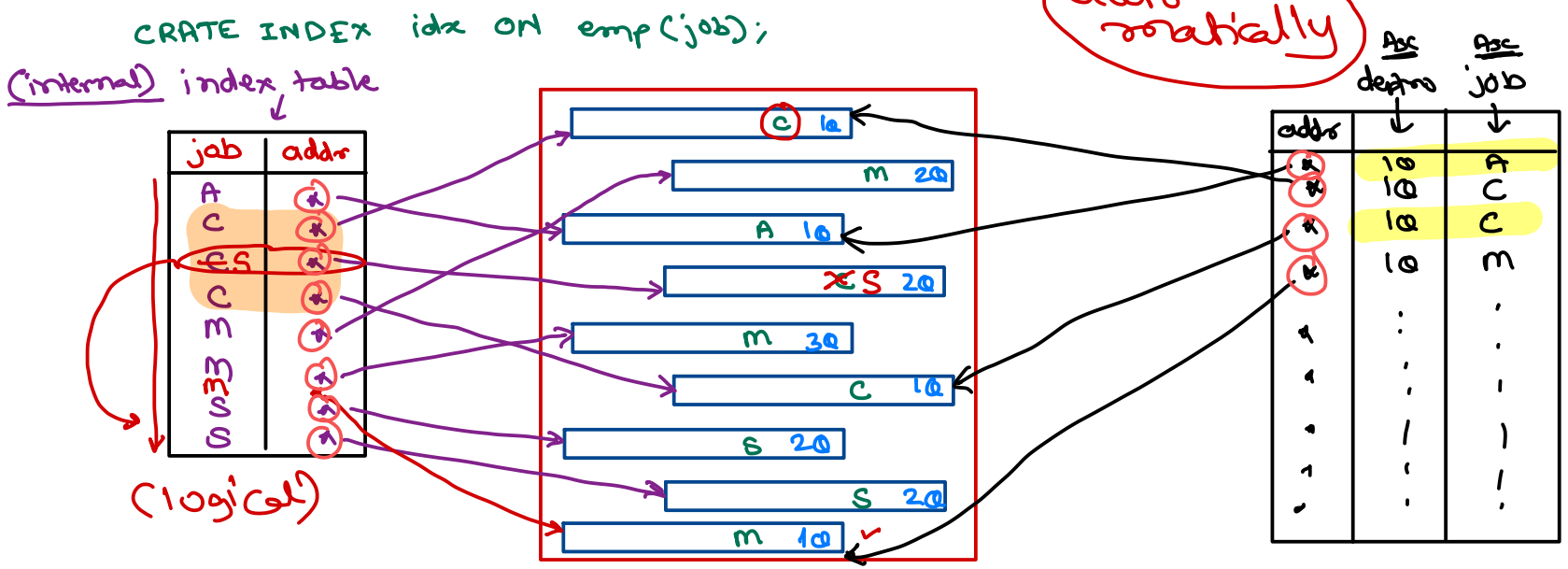
Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Index

- Index enable faster searching in tables by indexed columns.
  - CREATE INDEX idx_name ON table(column);
- One table can have multiple indexes on different columns/order.
- Typically indexes are stored as some data structure (like BTREE or HASH) on disk.
- Indexes are updated during DML operations. So DML operation are slower on indexed tables.



Index

- Index can be ASC or DESC.
  - It cause storage of key values in respective order (MySQL 8.x onwards).
  - ASC/DESC index is used by optimizer on ORDER BY queries.
- There are four types of indexes:
  - Simple index
    - CREATE INDEX idx_name ON table(column [ASC|DESC]);
  - Unique index
    - CREATE UNIQUE INDEX idx_name ON table(column [ASC|DESC]);
    - Doesn't allow duplicate values.
  - Composite index
    - CREATE INDEX idx_name ON table(column1 [ASC|DESC], column2 [ASC|DESC]);
    - Composite index can also be unique. Do not allow duplicate combination of columns.
  - Clustered index
    - PRIMARY index automatically created on Primary key for row lookup.
    - If primary key is not available, hidden index is created on synthetic column.
    - It is maintained in tabular form and its reference is used in other indexes.

Index

- Indexes should be created on shorter (INT, CHAR, …) columns to save disk space.
- Few RDBMS do not allow indexes on external columns i.e. TEXT, BLOB.
- MySQL support indexing on TEXT/BLOB up to n characters.
  - CREATE TABLE test (blob_col BLOB, …, INDEX(blob_col(10)));
- To list all indexes on table:
  - SHOW INDEXES ~~ON~~ FROM table;
- To drop an index:
  - DROP INDEX idx_name ON table;
- When table is dropped, all indexes are automatically dropped.
- Indexes should not be created on the columns not used frequent search, ordering or grouping operations.
- Columns in join operation should be indexed for better performance.

Query performance

- Few RDBMS features ensure better query performance.
  - Index speed up execution of SELECT queries (search operations).
  - Correlated sub-queries execute faster.
- Query performance can observed using EXPLAIN statement.
  - EXPLAIN FORMAT=JSON SELECT …;
- EXPLAIN statement shows
  - Query cost (Lower is the cost, faster is the query execution).
  - Execution plan (Algorithm used to execute query e.g. loop, semi-join, materialization, etc).
- Optimizations can be enabled or disabled by optimizer_switch system variable.
  - SELECT @@optimizer_switch;
  - SET @@optimizer_switch='materialization=off';

Constraints → DDL → Integrity/Validity of data.

- Constraints are restrictions imposed on columns. → restricts values to be added in the column.
- There are five constraints
  - NOT NULL → Col
  - UNIQUE → Col, Tbl
  - PRIMARY KEY → Col, Tbl
  - FOREIGN KEY → Col, Tbl
  - CHECK → Col, Tbl

  OR given while creating table. / given later using ALTER TABLE.
- Few constraints can be applied at either column level or table level. Few constraints can be applied on both.
- Optionally constraint names can be mentioned while creating the constraint. If not given, it is auto-generated.
- Each DML operation check the constraints before manipulating the values. If any constraint is violated, error is raised. → slower.

Constraints

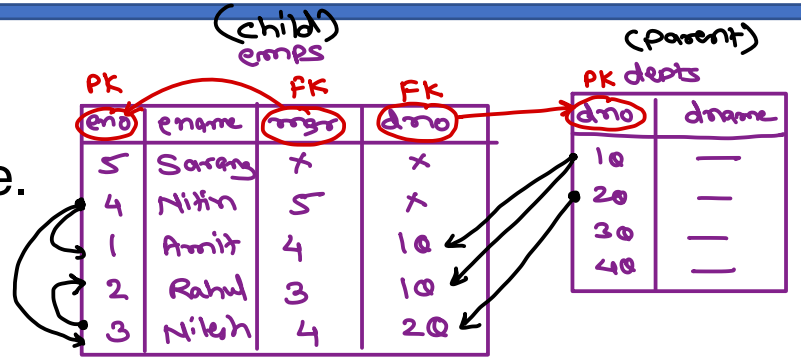
- NOT NULL
  - NULL values are not allowed.
  - Can be applied at column level only.
  - CREATE TABLE table(c1 TYPE NOT NULL, …);
- UNIQUE
  - Duplicate values are not allowed.
  - NULL values are allowed.
  - Not applicable for TEXT and BLOB.
  - UNIQUE can be applied on one or more columns.
  - Internally creates unique index on the column (fast searching).
  - Can be applied at column level or table level.
    - CREATE TABLE table(c1 TYPE UNIQUE, …);
    - CREATE TABLE table(c1 TYPE, …, UNIQUE(c1));
    - CREATE TABLE table(c1 TYPE, …, CONSTRAINT constraint_name UNIQUE(c1));

Constraints

- PRIMARY KEY
  - Column or set of columns that uniquely identifies a row.
  - Only one primary key is allowed for a table.
  - Primary key column cannot have duplicate or NULL values.
  - Internally index is created on PK column. → Primary Index (Clustered)
  - TEXT/BLOB cannot be primary key.
  - If no obvious choice available for PK, composite or surrogate PK can be created.
  - Creating PK for a table is a good practice.
  - PK can be created at table level or column level.
  - CREATE TABLE table(c1 TYPE PRIMARY KEY, …);
  - CREATE TABLE table(c1 TYPE, …, PRIMARY KEY(c1));
  - CREATE TABLE table(c1 TYPE, …, CONSTRAINT constraint_name PRIMARY KEY(c1));
  - CREATE TABLE table(c1 TYPE, c2 TYPE, …, PRIMARY KEY(c1, c2));

Constraints

- FOREIGN KEY
  - Column or set of columns that references a column of some table.
  - If column belongs to the same table, it is "self referencing".
  - Foreign key constraint is specified on child table column.
  - FK can have duplicate values as well as null values.
  - FK constraint is applied on column of child table (not on parent table).
  - Child rows cannot be deleted, until parent rows are deleted.
  - MySQL have ON DELETE CASCADE clause to ensure that child rows are automatically deleted, when parent row is deleted. ON UPDATE CASCADE clause does same for UPDATE operation.
  - By default foreign key checks are enabled. They can be disabled by
    - SET @@foreign_key_checks = 0;
  - FK constraint can be applied on table level as well as column level.
  - CREATE TABLE child(c1 TYPE, …, FOREIGN KEY (c1) REFERENCES parent(col))



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

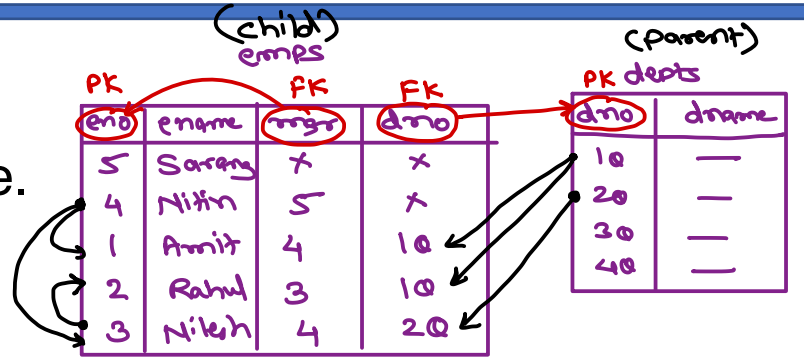
MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Constraints

Data Integrity

- FOREIGN KEY
  - Column or set of columns that references a column of some table.
  - If column belongs to the same table, it is "self referencing".
  - Foreign key constraint is specified on child table column.
  - FK can have duplicate values as well as null values.
  - FK constraint is applied on column of child table (not on parent table).
  - Child rows cannot be deleted, until parent rows are deleted.
  - MySQL have ON DELETE CASCADE clause to ensure that child rows are automatically deleted, when parent row is deleted. ON UPDATE CASCADE clause does same for UPDATE operation.
  - By default foreign key checks are enabled. They can be disabled by
    - SET @@foreign_key_checks = 0; → No checks will be performed while DML. So parent records can be inserted, update & deleted directly. This is done only in special cases like DB backup.
  - FK constraint can be applied on table level as well as column level.
  - CREATE TABLE child(c1 TYPE, …, FOREIGN KEY (c1) REFERENCES parent(col))



Constraints

- CHECK
  - CHECK is integrity constraint in SQL.
  - CHECK constraint specifies condition on column.
  - Data can be inserted/updated only if condition is true; otherwise error is raised.
  - CHECK constraint can be applied at table level or column level.
  - CREATE TABLE table(c1 TYPE, c2 TYPE CHECK condition1, …, CHECK condition2);

Sub queries

- Sub-query is query within query. Typically it work with SELECT statements.
- Output of inner query is used as input to outer query.
- If no optimization is enabled, for each row of outer query result, sub-query is executed once. This reduce performance of sub-query.
- Single row sub-query
  - Sub-query returns single row.
  - Usually it is compared in outer query using relational operators.

Sub queries

- Multi-row sub-query
  - Sub-query returns multiple rows.
  - Usually it is compared in outer query using operators like IN, ANY or ALL.
    - IN operator checks for equality with results from sub-queries.
    - ANY operator compares with one of the result from sub-queries.
    - ALL operator compares with all the results from sub-queries.

Sub queries

- Correlated sub-query
  - If number of results from sub-query are reduced, query performance will increase.
  - This can be done by adding criteria (WHERE clause) in sub-query based on outer query row.
  - Typically correlated sub-query use IN, ALL, ANY and EXISTS operators.

doesn't Compare inner query result with outer row.

it only checks if inner query is returning any row. (cnt > 0).

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Sub query

- Sub queries with UPDATE and DELETE are not supported in all RDBMS.
- In MySQL, Sub-queries in UPDATE/DELETE is allowed, but sub-query should not SELECT from the same table, on which UPDATE/DELETE operation is in progress.

Views

- RDBMS view represents view (projection) of the data.
- View is based on SELECT statement.
- Typically it is restricted view of the data (limited rows or columns) from one or more tables (joins and/or sub-queries) or summary of the data (grouping).
- Data of view is not stored on server hard-disk; but its SELECT statement is stored in compiled form. It speed up execution of view.

where

views are not materialized.

data is not copied / stored.

Views

- Views are of two types: Simple view and Complex view
- Usually if view contains computed columns, group by, joins or sub-queries, then the views are said to be complex. DML operations are not supported on these views.
- DML operations on Simple view affects underlying table. & vice versa
- View can be created with CHECK OPTION to ensure that DML operations can be performed only the data visible in that view.

View

- Views can be differentiated with: SHOW FULL TABLES.
- Views can be dropped with DROP VIEW statement.
- View can be based on another view.

* SHOW CREATE VIEW viewname ;
* Cannot create index on views.

- Applications of views
  - Security: Providing limited access to the data.
  - Hide source code of the table. → DESC tablename or SHOW CREATE TABLE ... ;
  - Simplifies complex queries.

Data Control Language

- Security is built-in feature of any RDBMS. It is implemented in terms of permissions (a.k.a. privileges).
- There are two types of privileges.
- System privileges
  - Privileges for certain commands i.e. CREATE TABLE, CREATE USER, CREATE TRIGGER, ...
  - Typically these privileges are given to the database administrator. (MySQL root login).
- Object privileges
  - RDBMS objects are table, view, stored procedure, function, triggers, …
  - Can perform operations on the objects i.e. INSERT, UPDATE, DELETE, SELECT, CALL, ...
  - Typically these privileges are given to the database users.

Database users, table struct & other system level info is maintained in system tables (in system dbs).

system dbs: mysql, sys, performance_schema, information_schema.

visible by root login: SHOW DATABASES;

Data Control Language

- Permissions are given to user using GRANT command.
  - GRANT CREATE TABLE TO user@host;
  - GRANT CREATE TABLE, CREATE VIEW TO user1@host, user2@host;
  - GRANT SELECT ON db.table TO user@host;
  - GRANT SELECT, INSERT, UPDATE ON db.table TO user@host;
  - GRANT ALL ON db.* TO user@host;
- By default one user cannot give permissions to other user. This can be enabled using WITH GRANT OPTION.
  - GRANT ALL ON *.* TO user@host WITH GRANT OPTION;
- Permissions for the user can be listed using SHOW GRANTS command.
- Permissions assigned to any user can be withdrawn using REVOKE command.
  - REVOKE SELECT, INSERT ON db.table FROM user@host;

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Data Control Language

- Security is built-in feature of any RDBMS. It is implemented in terms of permissions (a.k.a. privileges).
- There are two types of privileges.
- System privileges
  - Privileges for certain commands i.e. CREATE TABLE, CREATE USER, CREATE TRIGGER, ...
  - Typically these privileges are given to the database administrator. (MySQL root login).
- Object privileges
  - RDBMS objects are table, view, stored procedure, function, triggers, …
  - Can perform operations on the objects i.e. INSERT, UPDATE, DELETE, SELECT, CALL, ...
  - Typically these privileges are given to the database users.

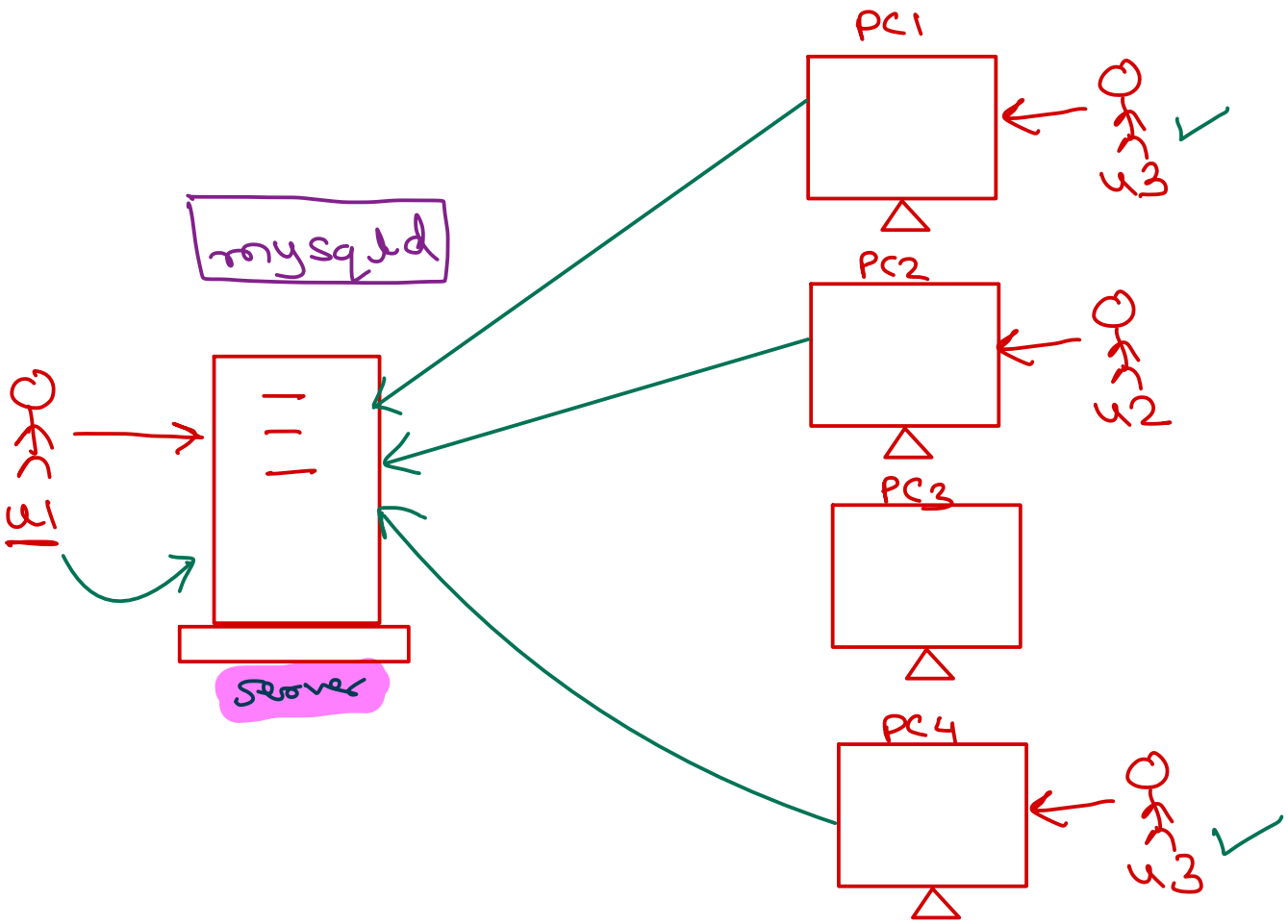
Database users, table struct & other system level info is maintained in system tables (in system dbs).

system dbs: mysql, sys, performance_schema, information_schema.

visible by root login: SHOW DATABASES;

Data Control Language

- Permissions are given to user using GRANT command.
  - GRANT CREATE TABLE TO user@host;
  - GRANT CREATE TABLE, CREATE VIEW TO user1@host, user2@host;
  - GRANT SELECT ON db.table TO user@host;
  - GRANT SELECT, INSERT, UPDATE ON db.table TO user@host;
  - GRANT ALL ON db.* TO user@host;
- By default one user cannot give permissions to other user. This can be enabled using WITH GRANT OPTION.
  - GRANT ALL ON *.* TO user@host WITH GRANT OPTION;
- Permissions for the user can be listed using SHOW GRANTS command.
- Permissions assigned to any user can be withdrawn using REVOKE command.
  - REVOKE SELECT, INSERT ON db.table FROM user@host;



Create user u1@localhost identified by 'u1';

Create user u2@pc2 identified by 'u2';

Create user u3@'%' identified by 'u3';

terminal> mysql → default localhost

-h server

-u user name

-p password

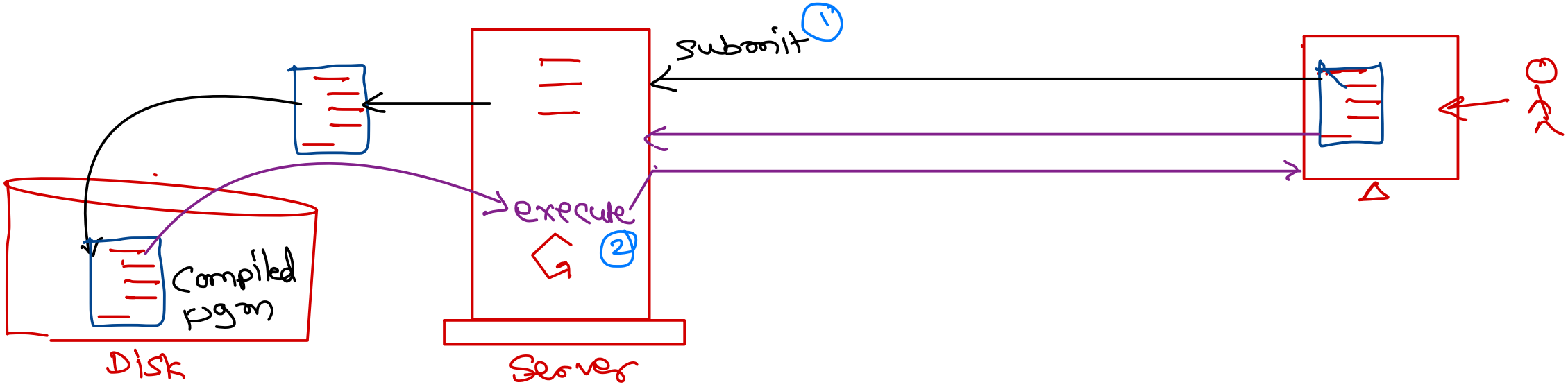
db name

MySQL Programming

- RDBMS Programming is an ISO standard – part of SQL standard – since 1992.
- SQL/PSM stands for Persistent Stored Module.
- Inspired from PL/SQL - Programming language of Oracle.
- PSM allows writing programs for RDBMS. The program contains set of SQL statements along with programming constructs e.g. variables, if-else, loops, case, ...
- PSM is a block language. Blocks can be nested into another block.
- MySQL program can be a stored procedure, function or trigger.

MySQL Programming

- MySQL PSM program is written by db user (programmers).
- It is submitted from client, server check syntax & store them into db in compiled form.
- The program can be executed by db user when needed.
- Since programs are stored on server in compiled form, their execution is very fast.
- All these programs will run in server memory.



Stored Procedure

- Stored Procedure is a routine. It contains multiple SQL statements along with programming constructs.
- Procedure doesn't return any value (like void fns in C).
- Procedures can take zero or more parameters.
- Procedures are created using CREATE PROCEDURE and deleted using DROP PROCEDURE.
- Procedures are invoked/called using CALL statement.
- Result of stored procedure can be
  - returned via OUT parameter.
  - inserted into another table.
  - produced using SELECT statement (at end of SP).
- Delimiter should be set before writing SQL query.

Stored Procedure

```
CREATE TABLE result(v1 DOUBLE, v2 VARCHAR(50));

DELIMITER $$

CREATE PROCEDURE sp_hello()
BEGIN
    INSERT INTO result VALUES(1, 'Hello World');
END;
$$

DELIMITER ;

CALL sp_hello();

SELECT * FROM result;
```

1
```
-- 01_hello.sql (using editor)
DROP PROCEDURE IF EXISTS sp_hello;
DELIMITER $$
CREATE PROCEDURE sp_hello()
BEGIN
    SELECT 1 AS v1, 'Hello World' AS v2;
END;
$$
DELIMITER ;
```

2
```
SOURCE /path/to/01_hello.sql
```

3
```
CALL sp_hello();
```

Stored Procedure – PSM Syntax

VARIABLES

```
DECLARE varname DATATYPE;
DECLARE varname DATATYPE DEFAULT init_value;
SET varname = new_value;
SELECT new_value INTO varname;
SELECT expr_or_col INTO varname FROM table_name;
```

PARAMETERS

```
CREATE PROCEDURE sp_name(PARAMTYPE p1 DATATYPE)
BEGIN
    …
END;

-- IN param: Initialized by calling program.
-- OUT param: Initialized by called procedure.
-- INOUT param: Initialized by calling program and
modified by called procedure
-- OUT & INOUT param declared as session variables.

CREATE PROCEDURE sp_name(OUT p1 INT)
BEGIN
    SELECT 1 INTO p1;
END;

SET @res = 0;
CALL sp_name(@res);
SELECT @res;
```

IF-ELSE

```
IF condition THEN
     body;
END IF;
-----------------------
IF condition THEN
     if-body;
ELSE
     else-body;
END IF;
-----------------------
IF condition THEN
     if1-body;
ELSE
     IF condition THEN
            if2-body;
     ELSE
            else2-body;
     END IF;
END IF;
-----------------------
IF condition THEN
     if1-body;
ELSEIF condition THEN
     if2-body;
ELSE
     else-body;
END IF;
```

LOOPS

```
WHILE condition DO
    body;
END WHILE;
------------------
REPEAT
    body;
UNTIL condition
END REPEAT;
------------------
label: LOOP
IF condition THEN
        ...
        LEAVE label;
    END IF;
    ...
END LOOP;
```

CASE-WHEN

```
CASE
WHEN condition THEN
      body;
WHEN condition THEN
      body;
ELSE
       body;
END CASE;
```

SHOW PROCEDURE

```
SHOW PROCEDURE STATUS
LIKE ‘sp_name’;

SHOW CREATE PROCEDURE sp_name;
```

DROP PROCEDURE

```
DROP PROCEDURE
IF EXISTS sp_name;
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Stored Procedure – PSM Syntax

VARIABLES

```
DECLARE varname DATATYPE;
DECLARE varname DATATYPE DEFAULT init_value;
SET varname = new_value;
SELECT new_value INTO varname;
SELECT expr_or_col INTO varname FROM table_name;
```

PARAMETERS

```
CREATE PROCEDURE sp_name(PARAMTYPE p1 DATATYPE)
BEGIN
    …
END;

-- IN param: Initialized by calling program.
-- OUT param: Initialized by called procedure.
-- INOUT param: Initialized by calling program and
modified by called procedure
-- OUT & INOUT param declared as session variables.

CREATE PROCEDURE sp_name(OUT p1 INT)
BEGIN
    SELECT 1 INTO p1;
END;

SET @res = 0;
CALL sp_name(@res);
SELECT @res;
```

mysql client

IF-ELSE

```
IF condition THEN
    body;
END IF;
```

```
IF condition THEN
    if-body;
ELSE
    else-body;
END IF;
```

```
IF condition THEN
    if1-body;
ELSE
    IF condition THEN
        if2-body;
    ELSE
        else2-body;
    END IF;
END IF;
```

```
IF condition THEN
    if1-body;
ELSEIF condition THEN
    if2-body;
ELSE
    else-body;
END IF;
```

LOOPS

loop
repeat

```
WHILE condition DO
    body;
END WHILE;
```

if cnd true.

---

like do-while

```
REPEAT
    body;
UNTIL condition
END REPEAT;
```

executed at least once.

repeat if cnd is false.

```
label: LOOP
IF condition THEN
        ...
        LEAVE label;
    END IF;
    ...
END LOOP;
```

like infinite loop (no condn).

like "break" keyword in C. exit the loop.

CASE-WHEN

```
CASE
WHEN condition THEN
        body;
WHEN condition THEN
        body;
ELSE
        body;
END CASE;
```

SHOW PROCEDURE

```
SHOW PROCEDURE STATUS
LIKE 'sp_name';
```

→ sp%

```
SHOW CREATE PROCEDURE sp_name;
```

DROP PROCEDURE

```
DROP PROCEDURE
IF EXISTS sp_name;
```

MySQL Stored Functions

- Stored Functions are MySQL programs like stored procedures.
- Functions can be having one or more parameters. MySQL allows only IN params.
- Functions must return some value using RETURN statement.
- Function entire code is stored in system table. (like procedure)
- Like procedures, functions allows statements like local variable declarations, if-else, case, loops, etc. One function can invoke another function/procedure and vice-versa. The functions can also be recursive.
- There are two types of functions: DETERMINISTIC and NOT DETERMINISTIC.

CREATE FUNCTION

```
CREATE FUNCTION fn_name(p1 TYPE)
RETURNS TYPE
[NOT] DETERMINISTIC
BEGIN
   body;
   RETURN value;
END;
```

SHOW FUNCTION

```
SHOW FUNCTION STATUS LIKE 'fn_name';

SHOW CREATE FUNCTION fn_name;
```

DROP FUNCTION

```
DROP FUNCTION IF EXISTS fn_name;
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Stored Procedure – PSM Syntax

VARIABLES

```
DECLARE varname DATATYPE;
DECLARE varname DATATYPE DEFAULT init_value;
SET varname = new_value;
SELECT new_value INTO varname;
SELECT expr_or_col INTO varname FROM table_name;
```

PARAMETERS

```
CREATE PROCEDURE sp_name(PARAMTYPE p1 DATATYPE)
BEGIN
    …
END;

-- IN param: Initialized by calling program.
-- OUT param: Initialized by called procedure.
-- INOUT param: Initialized by calling program and
modified by called procedure
-- OUT & INOUT param declared as session variables.

CREATE PROCEDURE sp_name(OUT p1 INT)
BEGIN
    SELECT 1 INTO p1;
END;

SET @res = 0;
CALL sp_name(@res);
SELECT @res;
```

mysql client

IF-ELSE

```
IF condition THEN
    body;
END IF;
```

```
IF condition THEN
    if-body;
ELSE
    else-body;
END IF;
```

```
IF condition THEN
    if1-body;
ELSE
    IF condition THEN
        if2-body;
    ELSE
        else2-body;
    END IF;
END IF;
```

```
IF condition THEN
    if1-body;
ELSEIF condition THEN
    if2-body;
ELSE
    else-body;
END IF;
```

LOOPS

loop
repeat

```
WHILE condition DO
    body;
END WHILE;
```

if cnd true.

like do-while executed at least once.

```
REPEAT
    body;
UNTIL condition
END REPEAT;
```

repeat if cnd is false.

```
label: LOOP
IF condition THEN
        ...
    LEAVE label;
    END IF;
    ...
END LOOP;
```

like infinite loop (no condn).

like "break" keyword in C. exit the loop.

ITERATE keyword is like "continue" in C.
skip further statements in the loop & continue executing next iteration of loop.

CASE-WHEN

```
CASE
WHEN condition THEN
    body;
WHEN condition THEN
    body;
ELSE
    body;
END CASE;
```

SHOW PROCEDURE

```
SHOW PROCEDURE STATUS
LIKE 'sp_name';
```

→ sp%

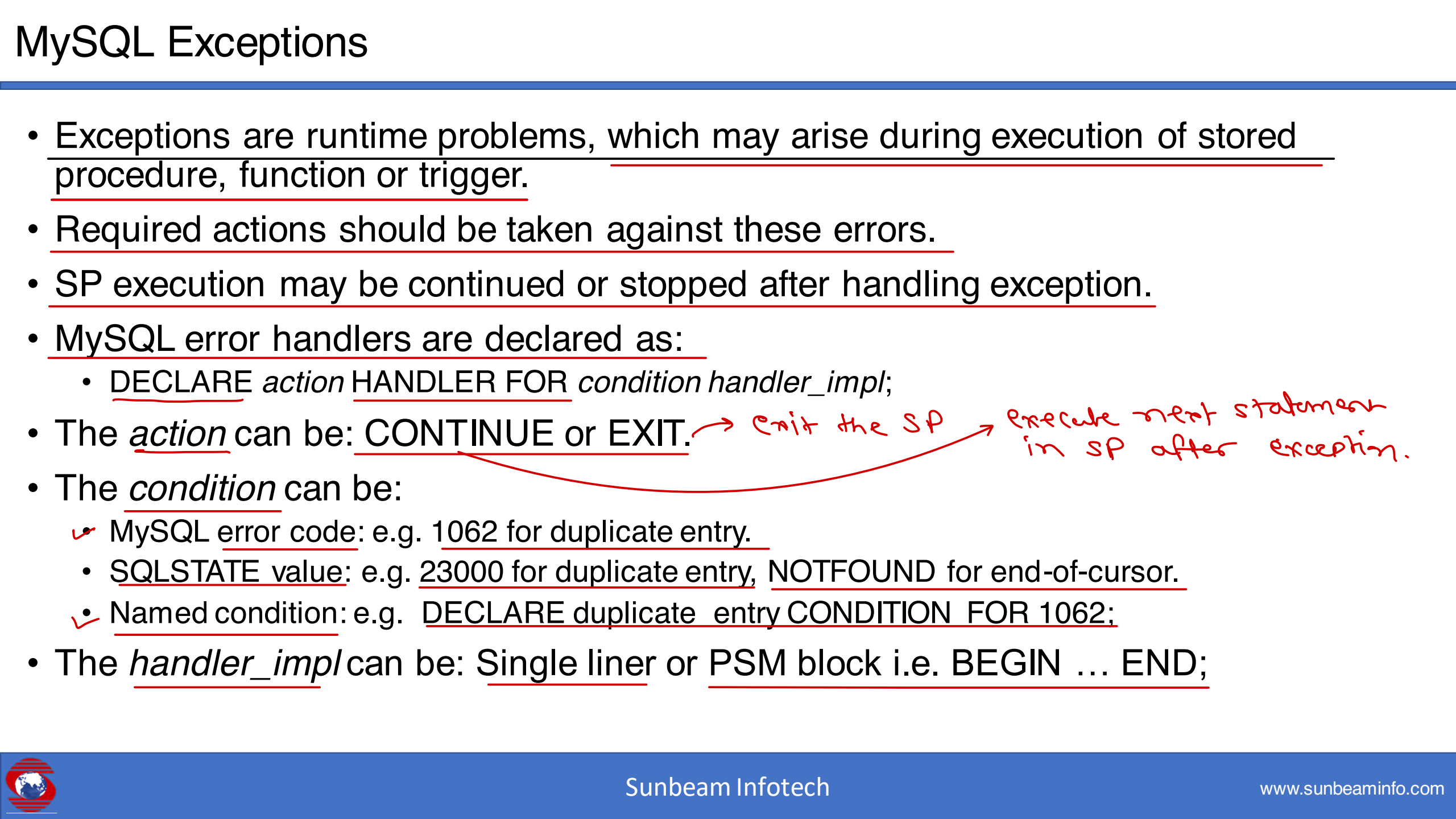
```
SHOW CREATE PROCEDURE sp_name;
```

DROP PROCEDURE

```
DROP PROCEDURE
IF EXISTS sp_name;
```

MySQL Exceptions

- Exceptions are runtime problems, which may arise during execution of stored procedure, function or trigger.
- Required actions should be taken against these errors.
- SP execution may be continued or stopped after handling exception.
- MySQL error handlers are declared as:
  - DECLARE *action* HANDLER FOR *condition handler_impl*;
- The *action* can be: CONTINUE or EXIT.
- The *condition* can be:
  - MySQL error code: e.g. 1062 for duplicate entry.
  - SQLSTATE value: e.g. 23000 for duplicate entry, NOTFOUND for end-of-cursor.
  - Named condition: e.g. DECLARE duplicate_entry CONDITION FOR 1062;
- The *handler_impl* can be: Single liner or PSM block i.e. BEGIN … END;



DECLARE v_cur CURSOR FOR SELECT * FROM dept;

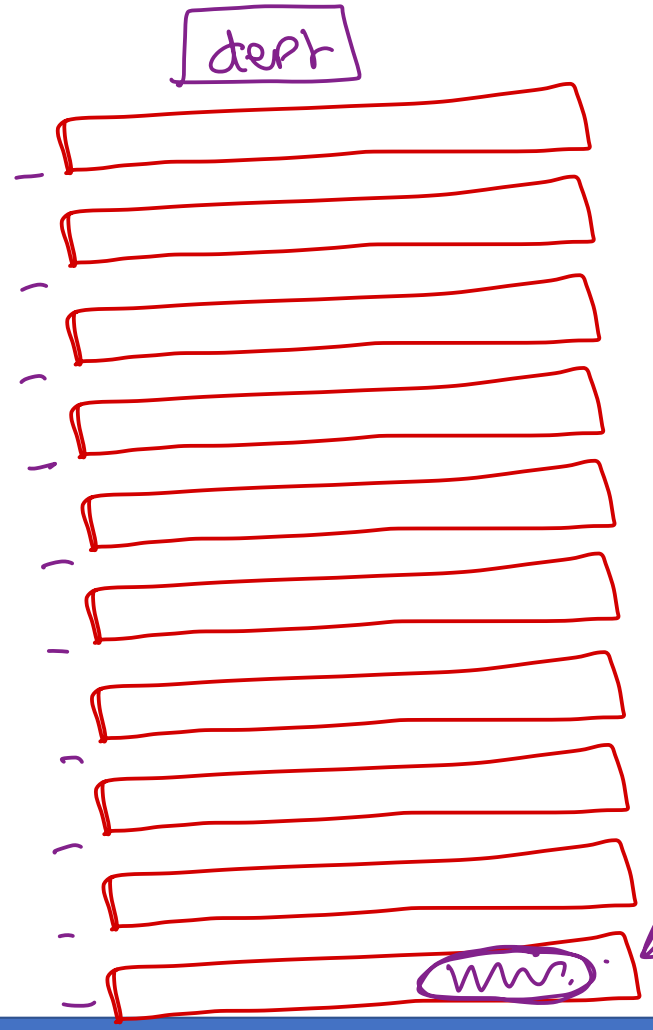
OPEN v_cur; ✓

LOOP

FETCH ... ✓

?

END LOOP;



Client 2

UPDATE dept

SET ...

WHERE ...

MySQL Stored Functions

- Stored Functions are MySQL programs like stored procedures.
- Functions can be having one or more parameters. MySQL allows only IN params.
- Functions must return some value using RETURN statement.
- Function entire code is stored in system table. (like procedure)
- Like procedures, functions allows statements like local variable declarations, if-else, case, loops, etc. One function can invoke another function/procedure and vice-versa. The functions can also be recursive.
- There are two types of functions: DETERMINISTIC and NOT DETERMINISTIC.

CREATE FUNCTION

```
CREATE FUNCTION fn_name(p1 TYPE)
RETURNS TYPE
[NOT] DETERMINISTIC
BEGIN
    body;
    RETURN value;
END;
```

SHOW FUNCTION

```
SHOW FUNCTION STATUS LIKE 'fn_name';

SHOW CREATE FUNCTION fn_name;
```

DROP FUNCTION

```
DROP FUNCTION IF EXISTS fn_name;
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

MySQL Triggers

- Triggers are supported by all standard RDBMS like Oracle, MySQL, etc.
- Triggers are not supported by WEAK RDBMS like MS-Access, SQLite.
- Triggers are not called by client's directly, so they don't have args & return value.
- Trigger execution is caused by DML operations on database.
  - BEFORE/AFTER INSERT, BEFORE/AFTER UPDATE, BEFORE/AFTER DELETE.
- Like SP/FN, Triggers may contain SQL statements with programming constructs. They may also call other SP or FN.
- However COMMIT/ROLLBACK is not allowed in triggers. They are executed in same transaction in which DML query is executed.

CREATE TRIGGER

```
CREATE TRIGGER trig_name
AFTER|BEFORE dml_op ON table
FOR EACH ROW
BEGIN
   body;
   -- use OLD & NEW keywords
   -- to access old/new rows.
   -- INSERT triggers - NEW rows.
   -- DELETE triggers - OLD rows.
END;
```

UPDATE triggers

SHOW TRIGGERS

```
SHOW TRIGGERS FROM db_name;
```

DROP TRIGGER

```
DROP TRIGGER trig_name;
```

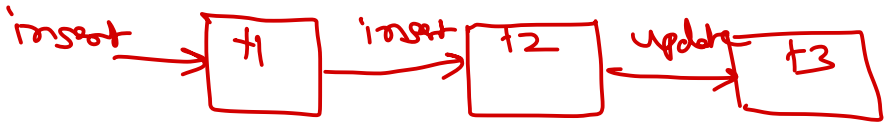
MySQL Triggers

- Applications of triggers:
  - Maintain logs of DML operations (Audit Trails).
  - Data cleansing before insert or update data into table. (Modify NEW value).
  - Copying each record AFTER INSERT into another table to maintain "Shadow table".
  - Copying each record AFTER DELETE into another table to maintain "History table".
  - Auto operations of related tables using cascading triggers.

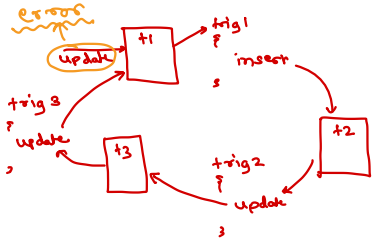
CHECK constraint for checking value.

SET NEW.ename = UPPER(OLD.ename);

emp → sal_history



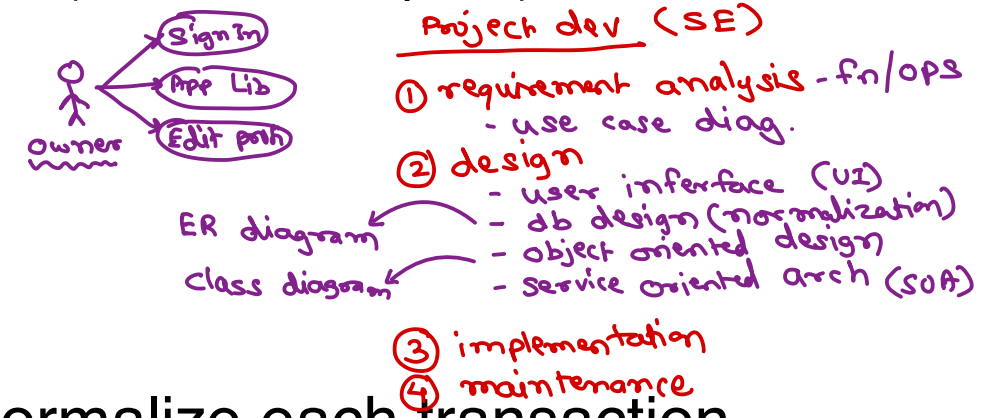
- Cascading triggers
  - One trigger causes execution of 2nd trigger, 2nd trigger causes execution of 3rd trigger and so on.
  - In MySQL, there is no upper limit on number of levels of cascading.
  - This is helpful in complicated business processes.



- Mutating table error
  - If cascading trigger causes one of the earlier trigger to re-execute, "mutating table" error is raised.
  - This prevents infinite loop and also rollback the current transaction.

Normalization

- Concept of table design: Table, Structure, Data Types, Width, Constraints, Relations.
- Goals:
  - Efficient table structure.
  - Avoid data redundancy i.e. unnecessary duplication of data (to save disk space).
  - Reduce problems of insert, update & delete.
- Done from input perspective.
- Based on user requirements.
- Part of software design phase.
- View entire appln on per transaction basis & then normalize each transaction separately.
- Transaction Examples:
  - Banking, Rail Reservation, Online Shopping.



Normalization

- For given transaction make list of all the fields.
- Strive for atomicity.
- Get general description of all field properties.
- For all practical purposes we can have a single table with all the columns. Give meaningful names to the table.
- Assign datatypes and widths to all columns on the basis of general desc of fields properties.
- Remove computed columns.
- Assign primary key to the table.
- At this stage data is in un-normalized form.
- UNF is starting point of normalization.

Thank you!

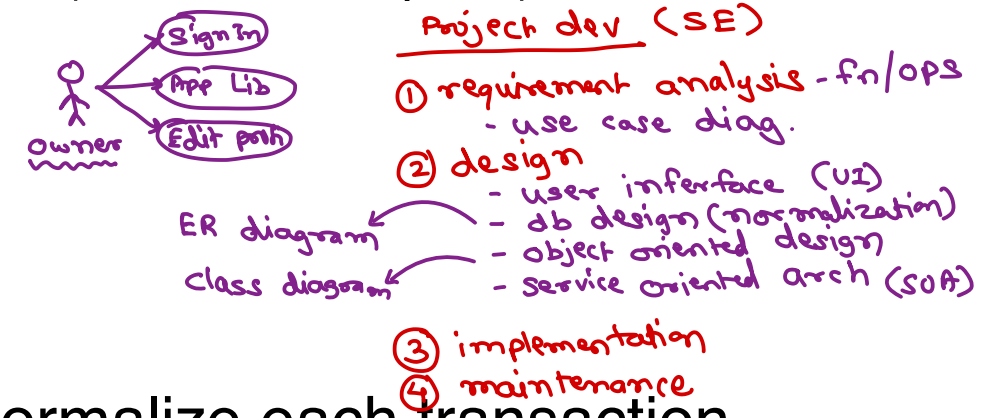
Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

Normalization

- Concept of table design: Table, Structure, Data Types, Width, Constraints, Relations.
- Goals:
  - Efficient table structure.
  - Avoid data redundancy i.e. unnecessary duplication of data (to save disk space).
  - Reduce problems of insert, update & delete.
- Done from input perspective.
- Based on user requirements.
- Part of software design phase.
- View entire appln on per transaction basis & then normalize each transaction separately.
- Transaction Examples:
  - Banking, Rail Reservation, Online Shopping.



Normalization

- For given transaction make list of all the fields.
- Strive for atomicity.
- Get general description of all field properties.
- For all practical purposes we can have a single table with all the columns. Give meaningful names to the table.
- Assign datatypes and widths to all columns on the basis of general desc of fields properties.
- Remove computed columns.
- Assign primary key to the table.
- At this stage data is in un-normalized form.
- UNF is starting point of normalization.

Normalization

- UNF suffers from
  - Insert anomaly → some data need to be inserted repeatedly
  - Update anomaly → same changes might be done at multiple records.
  - Delete anomaly → some of undesired data may be deleted.

1-NF → 25%.
steps 1-3

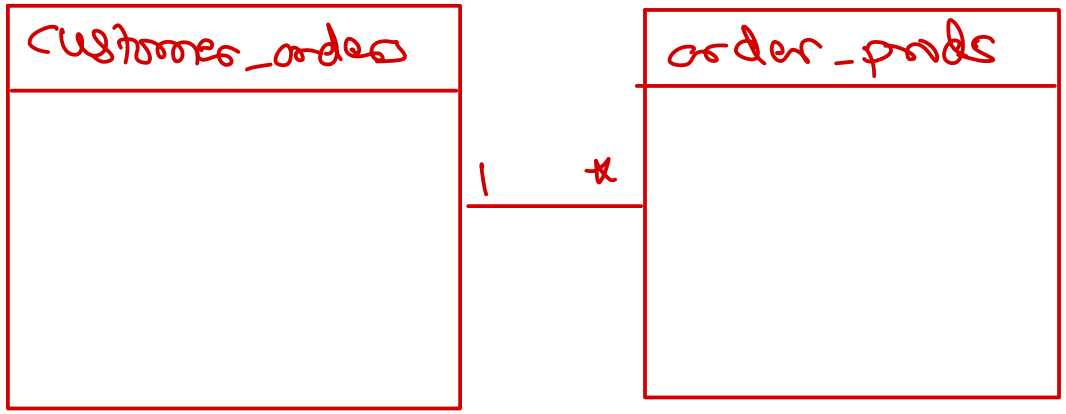
2-NF → 50%.
steps 4-6

3-NF → 25%.
steps 7-9

BCNF → optional
steps 10

Normalization 1-NF

- 1. Remove repeating group into a new table.
- 2. Key elements will be PK of new table.
- 3. (Optional) Add PK of original table to new table to give us Composite PK.
  - Repeat steps 1-3 infinitely -- to remove all repeating groups into new tables.
  - This is **1-NF**. No repeating groups present here. One to Many relationship between two tables.



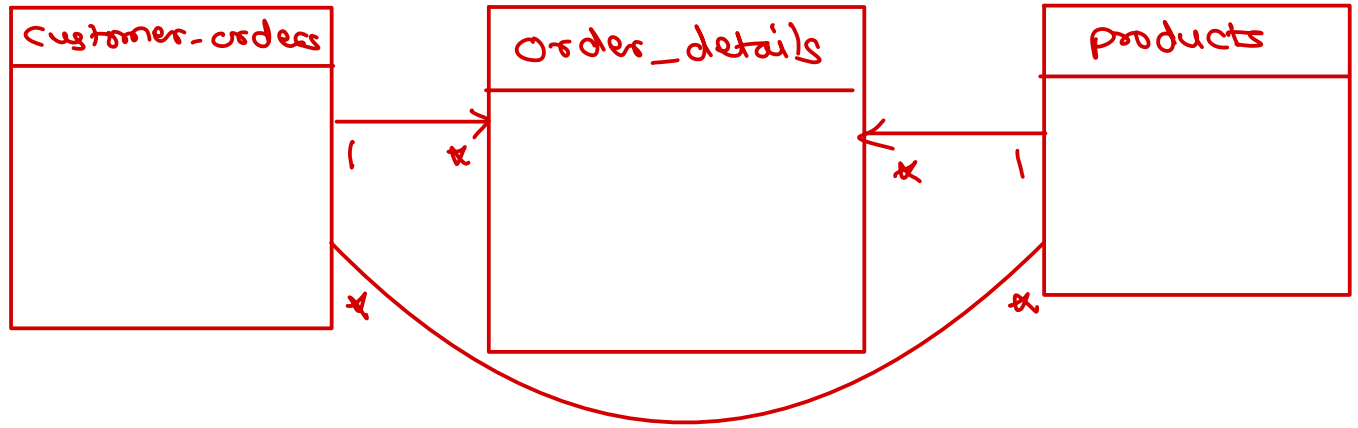
Normalization - 2 NF

- 4. Only table with composite PK to be examined.
- 5. Those columns that are not dependent on the entire composite PK, they are to be removed into a new table.
- 6. The key elements on which the non-key elements were originally dependent, it is to be added to the new table, and it will be the PK of new table.
  - Repeat steps 4-6 infinitely -- to separate all non-key elements from all tables with composite primary key.
  - This is **2-NF**. Many-to-Many relationship.

Many to many

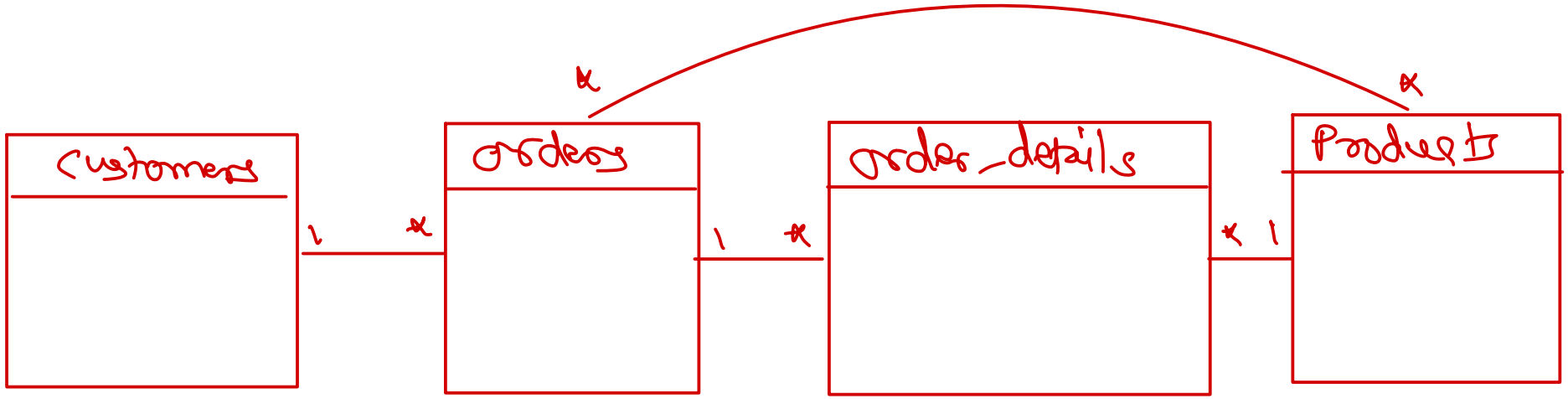
One order has many products.

One product is purchased in many orders.



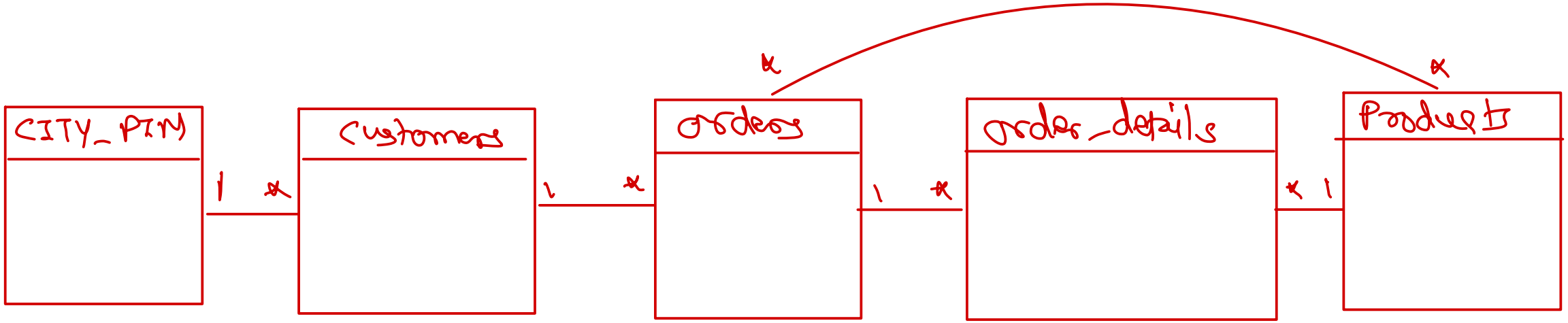
Normalization

- 7. Only non-key elements are examined for inter-dependencies.
- 8. Inter-dependent cols that are not directly related to PK, they are to be removed into a new table.
- 9. (a) Key ele will be PK of new table.
- 9. (b) The PK of new table is to be retained in original table for relationship purposes.
  - Repeat steps 7-9 infinitely to examine all non-key eles from all tables and separate them into new table if not dependent on PK.
  - This is **3-NF**.



Normalization

- To ensure data consistency (no wrong data entered by end user).
- Separate table to be created of well-known data. So that min data will be entered by the end user.
- This is BCNF or 4-NF.



SQL Keys

🔑 **An SQL key is either a single column (or attribute) or a group of columns that can uniquely identify rows (or tuples) in a table.**

🔑 **Super key is a single key or a group of multiple keys that can uniquely identify tuples in a table.**

🔑 **Candidate key is a single key or a group of multiple keys that uniquely identify rows in a table.**

🔑 **Primary key is the Candidate key selected by the database administrator to uniquely identify tuples in a table.**

🔑 **Alternate keys are those candidate keys which are not the Primary key.**

🔑 **Foreign key is an attribute which is a Primary key in its parent table, but is included as an attribute in another host table.**

Id	Name	Gender	City	Email	Dep_Id
1	Ajay	M	Delhi	ajay@gmail.com	1
2	Vijay	M	Mumbai	vijay@gmail.com	2
3	Radhika	F	Bhopal	radhika@gmail.com	1
4	Shikha	F	Jaipur	shikha@gmail.com	2
5	Hritik	M	Jaipur	hritik@gmail.com	2

5 rows in set (0.00 sec)

De-normalization

- Normalization will yield a structure that is non-redundant.
- Having too many inter-related tables will lead to complex and inefficient queries.
- To ensure better performance of analytical queries, few rules of normalization can be compromised.
- This process is de-normalization.

Codd's rules

- Proposed by Edgar F. Codd – pioneer of the RDBMS – in 1980.
- If any DBMS follow these rules, it can be considered as RDBMS.

- The 0th rule is the main rule known as "The foundation rule".
  - For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

- The rest of rules can be considered as elaboration of this foundation rule.

Codd's rules

- Rule 1: The information rule:
  - All information in a relational data base is represented explicitly at the logical level and in exactly one way – by values in tables.

- Rule 2: The guaranteed access rule:
  - Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

- Rule 3: Systematic treatment of null values:
  - Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

Codd's rules

metadata/structure/ psm

- Rule 4: Dynamic online catalog based on the relational model:
  - The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

MySQL 5.0+ → db information_schema.
TRIGGERS ROUTINES ...

- Rule 5: The comprehensive data sublanguage rule:
  - A relational system may support several languages. However, there must be at least one language that supports all functionalities of a RDBMS i.e. data definition, data manipulation, integrity constraints, transaction management, authorization.

SQL

Codd's rules

simple view → updateable.

- Rule 6: The view updating rule:
  - All views that are theoretically updatable are also updatable by the system.

view → group by → not updatable ×

- Rule 7: Possible for high-level insert, update, and delete:
  - The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.

→ bulk DML

- Rule 8: Physical data independence:
  - Application programs and terminal activities remain logically unbroken whenever any changes are made in either storage representations or access methods.

→ not fully implemented in MySQL.
- not all query work in all engines.

→ storage engine.

- Rule 9: Logical data independence:
  - Application programs & terminal activities remain logically unbroken when information-preserving changes of any kind that theoretically permit un-impairment are made to the base tables.

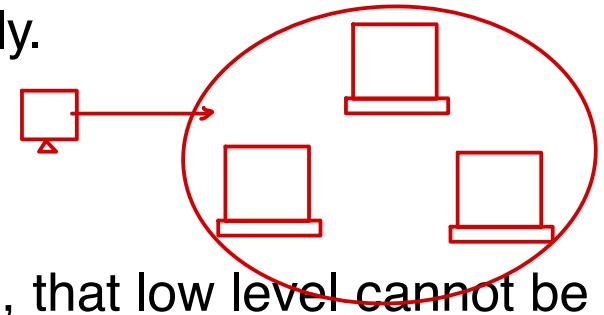
→ can be implemented using views.

Codd's rules

- Rule 10: Integrity independence:
  - Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

① PK cannot be null.
② child row (FK) cannot be added if parent row (PK) is not avail.

- Rule 11: Distribution independence:
  - The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.



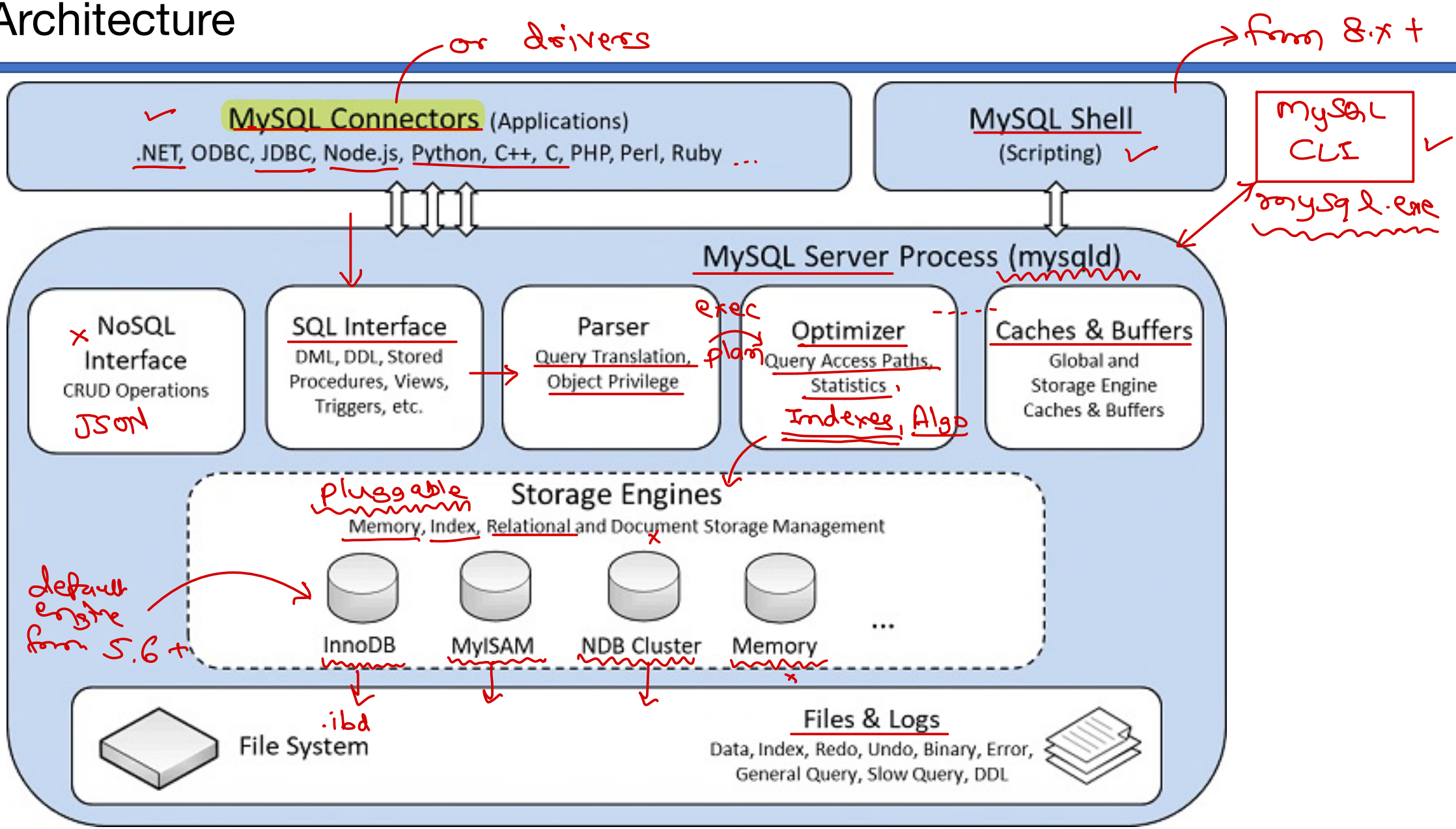
- Rule 12: The non-subversion rule:
  - If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).

→ no backdoor entry.

Temporary Tables

* Like views, but are materialized (stored in memory).
* faster query execution.
* limited to current session.

MySQL Architecture



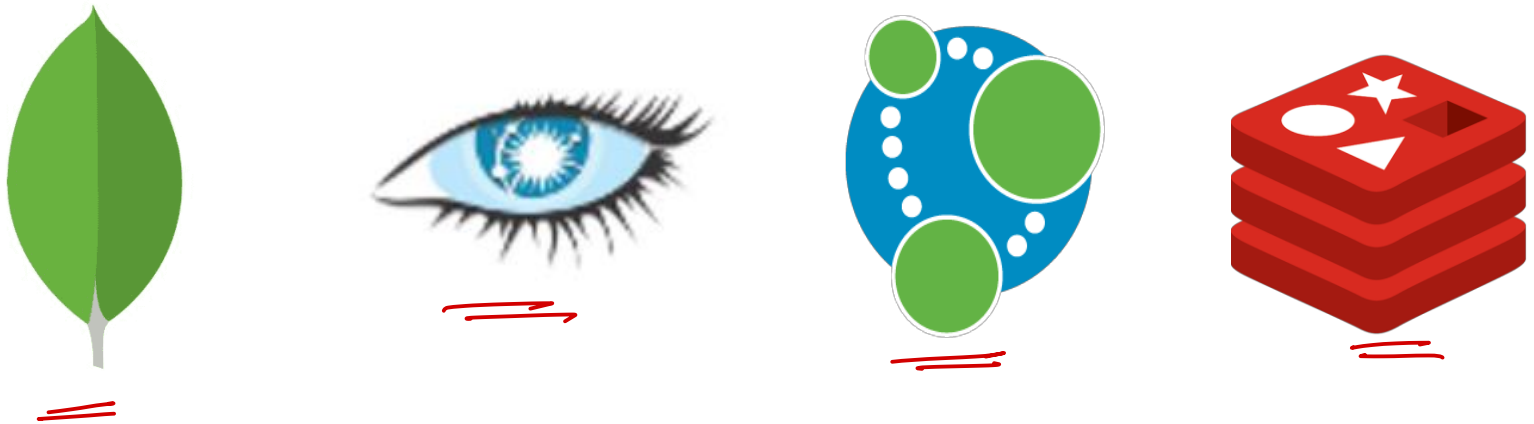
InnoDB vs MyISAM

if table is indexed.

- InnoDB has row-level locking. MyISAM only has full table-level locking.
- InnoDB has what is called referential integrity which involves supporting foreign keys (RDBMS) and relationship constraints, MyISAM does not (DBMS).
- InnoDB supports transactions, which means you can commit and roll back. MyISAM does not.
- InnoDB is more reliable as it uses transactional logs for auto recovery. MyISAM does not.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>



NoSQL Databases

Trainer: Mr. Nilesh Ghule

Agenda

- Introduction
- RDBMS vs NoSQL
- Scaling
- CAP theorem
- Advantages
- Limitations
- Applications
- Key-value database
- Column-oriented database
- Graph database
- Document database

Database

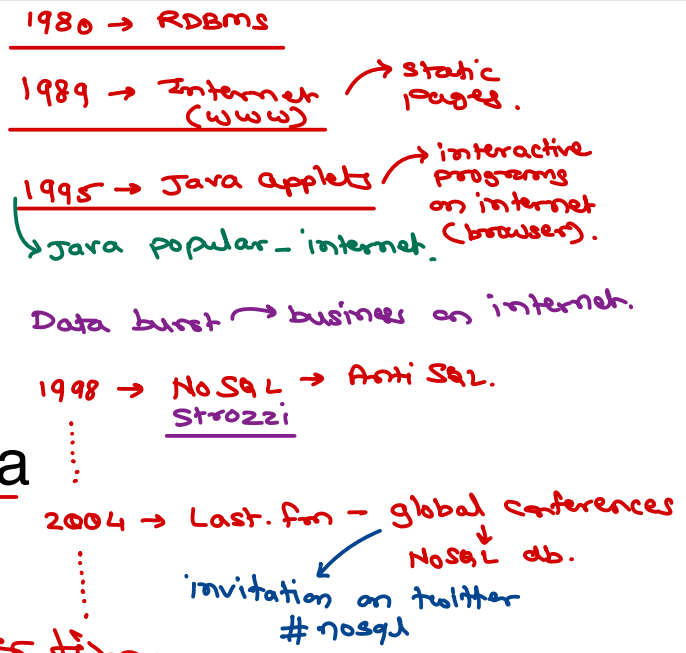
- A database is an organized collection of data, generally stored and accessed electronically from a computer system
- A database refers to a set of related data and the way it is organized
- Database Management System → CRUD ops
  - Software that allows users to interact with one or more databases and provides access to all of the data contained in the database
- Types
  - RDBMS → SQL lang → oracle, ms-sql, mysql, ...
  - NoSQL → Mongo, Cassandra, neo 4j, Hbase, Redis, ...
  - NewSQL → performance like NoSQL but interface like RDBMS.
    - HarperDb, Voltdb, ...

RDBMS

- The idea of RDBMS was borne in 1970 by E. F. Codd.
- Structured and organized data
- Structured query language (SQL)
- DML, DQL, DDL, DTL, DCL.
- Data and its relationships are stored in separate tables.
- Tight Consistency
- Based on Codd's rules
- ACID transactions.
  - Atomic
  - Consistent
  - Isolated
  - Durable

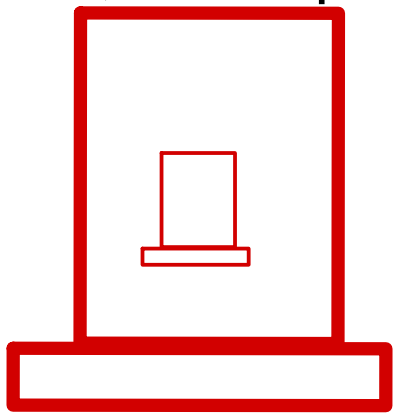
NoSQL

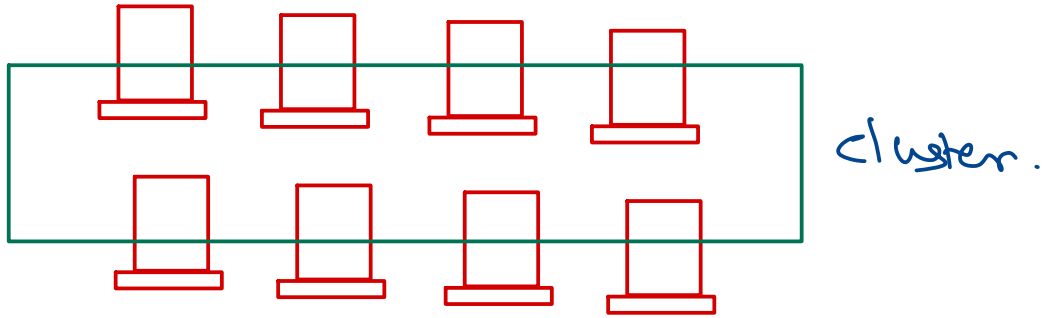
- Refer to non-relational databases
- Stands for Not Only SQL
- Term NoSQL was first used by Carlo Strozzi in 1998.
- No declarative query language
- No predefined schema, Unstructured and unpredictable data
- Eventual consistency rather ACID property
- Based on CAP Theorem
- Prioritizes high performance, high availability and scalability
- BASE Transaction
  - Basically Available
  - Soft state
  - Eventual consistency



Scaling

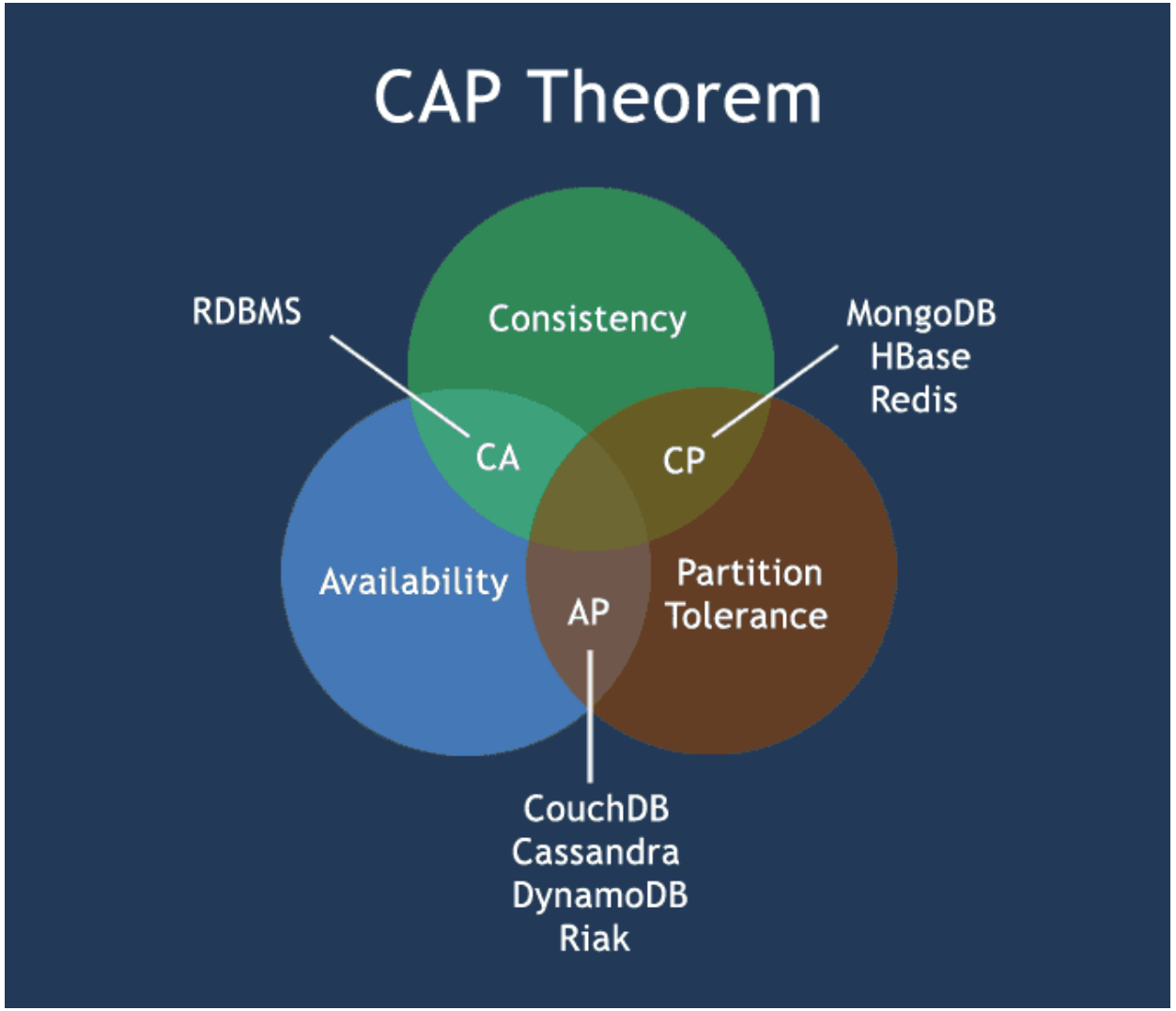
- Scalability is the ability of a system to expand to meet your business needs.
- E.g. scaling a web app is to allow more people to use your application.
- Types of scaling
  - Vertical scaling: Add resources within the same logical unit to increase capacity. E.g. add CPUs to an existing server, increase memory in the system or expanding storage by adding hard drives.
  - Horizontal scaling: Add more nodes to a system. E.g. adding a new computer to a distributed software application. Based on principle of distributed computing.
- NoSQL databases are designed for Horizontal scaling. So they are reliable, fault tolerant, better performance (at lower cost), speed.





CAP (Brewer's) Theorem

- **Consistency** - Data is consistent after operation. After an update operation, all clients see the same data.
- **Availability** - System is always on (i.e. service guarantee), no downtime.
- **Partition Tolerance** - System continues to function even the communication among the servers is unreliable.
- Brewer's Theorem
  - It is impossible for a distributed data store to simultaneously provide more than two out of the above three guarantees.



Advantages of NoSQL

- High scalability
  - This scaling up approach fails when the transaction rates and fast response requirements increase. In contrast to this, the new generation of NoSQL databases is designed to scale out (i.e. to expand horizontally using low-end commodity servers).
- Manageability and administration
  - NoSQL databases are designed to mostly work with automated repairs, distributed data, and simpler data models, leading to low manageability and administration.
- Low cost
  - NoSQL databases are typically designed to work with a cluster of cheap commodity servers, enabling the users to store and process more data at a low cost.
- Flexible data models
  - NoSQL databases have a very flexible data model, enabling them to work with any type of data; they don't comply with the rigid RDBMS data models. As a result, any application changes that involve updating the database schema can be easily implemented.

Disadvantages of NoSQL

- Maturity
  - Most NoSQL databases are pre-production versions with key features that are still to be implemented. Thus, when deciding on a NoSQL database, you should analyse the product properly to ensure the features are fully implemented and not still on the To-do list.
- Support
  - Support is one limitation that you need to consider. Most NoSQL databases are from start-ups which were open sourced. As a result, support is very minimal as compared to the enterprise software companies and may not have global reach or support resources.
- Limited Query Capabilities
  - Since NoSQL databases are generally developed to meet the scaling requirement of the web-scale applications, they provide limited querying capabilities. A simple querying requirement may involve significant programming expertise.
- Administration
  - Although NoSQL is designed to provide a no-admin solution, it still requires skill and effort for installing and maintaining the solution.
- Expertise
  - Since NoSQL is an evolving area, expertise on the technology is limited in the developer and administrator community.

RDBMS vs NoSQL

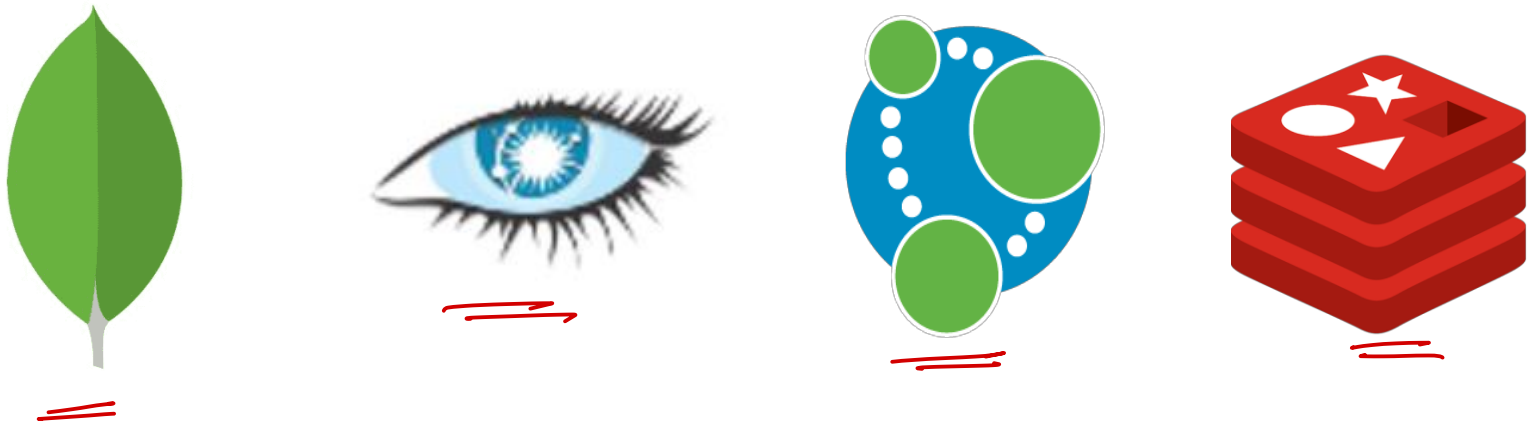
	RDBMS	NoSQL
Types	All types support SQL standard	Multiple types exists, such as document stores, key value stores, column databases, etc
History	Developed in 1970	Developed in 2000s
Examples	SQL Server, Oracle, MySQL	MongoDB, HBase, Cassandra, Redis, Neo4J
Data Storage Model	Data is stored in rows and columns in a table, where each column is of a specific type	The data model depends on the database type. It could be Key-value pairs, documents etc
Schemas	Fixed structure and schema	Dynamic schema. Structures can be accommodated
Scalability	Scale up approach is used	Scale out approach is used
Transactions	Supports ACID and transactions	Supports partitioning and availability
Consistency	Strong consistency	Dependent on the product [Eventual Consistency]
Support	High level of enterprise support	Open source model
Maturity	Have been around for a long time	Some of them are mature; others are evolving

NoSQL database

- NoSQL databases are non-relational.
- There is no standardization/rules of how NoSQL database to be designed.
- All available NoSQL databases can be broadly categorized as follows:
  - Key-value databases
  - Column-oriented databases
  - Graph databases
  - Document oriented databases

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

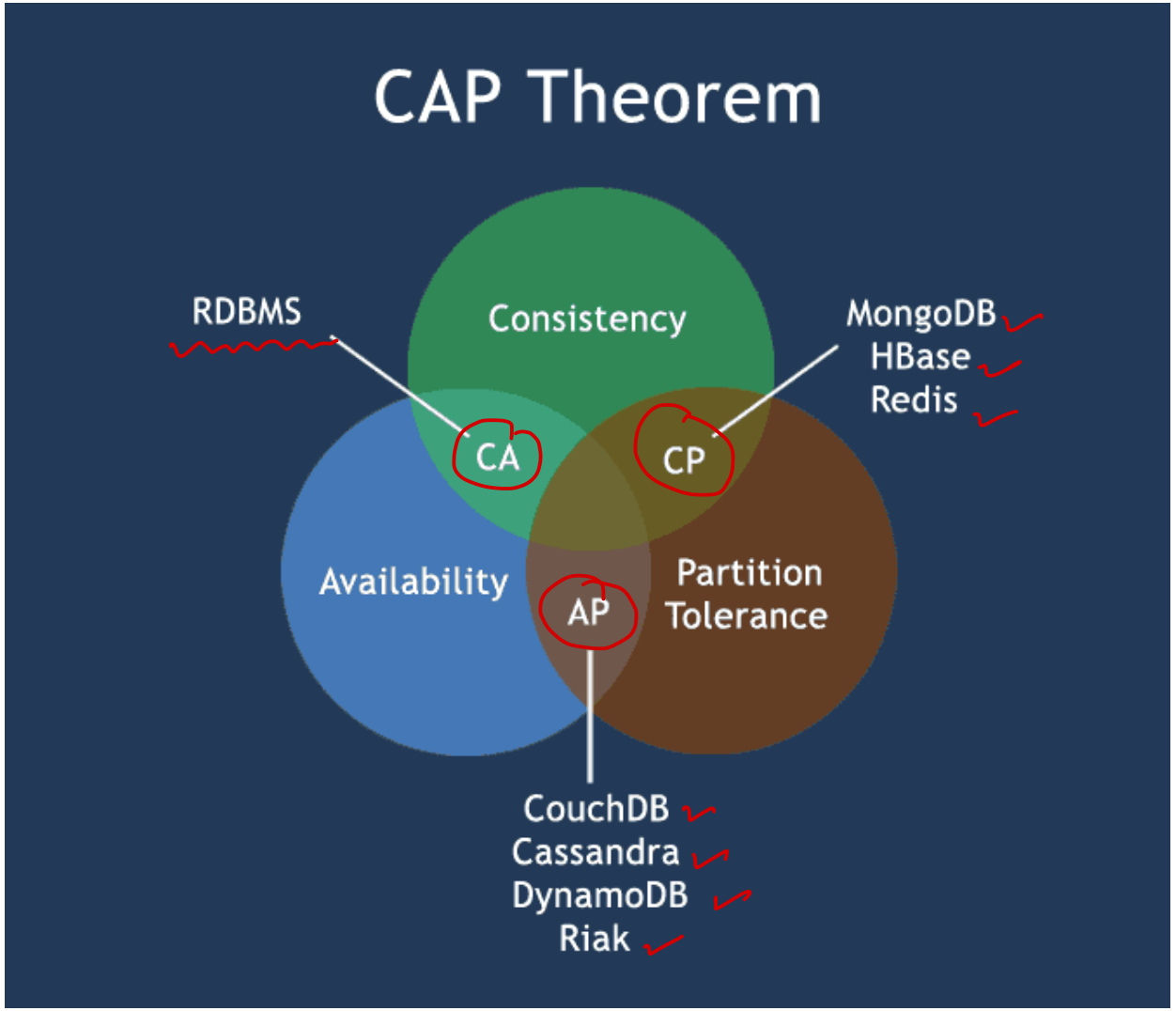


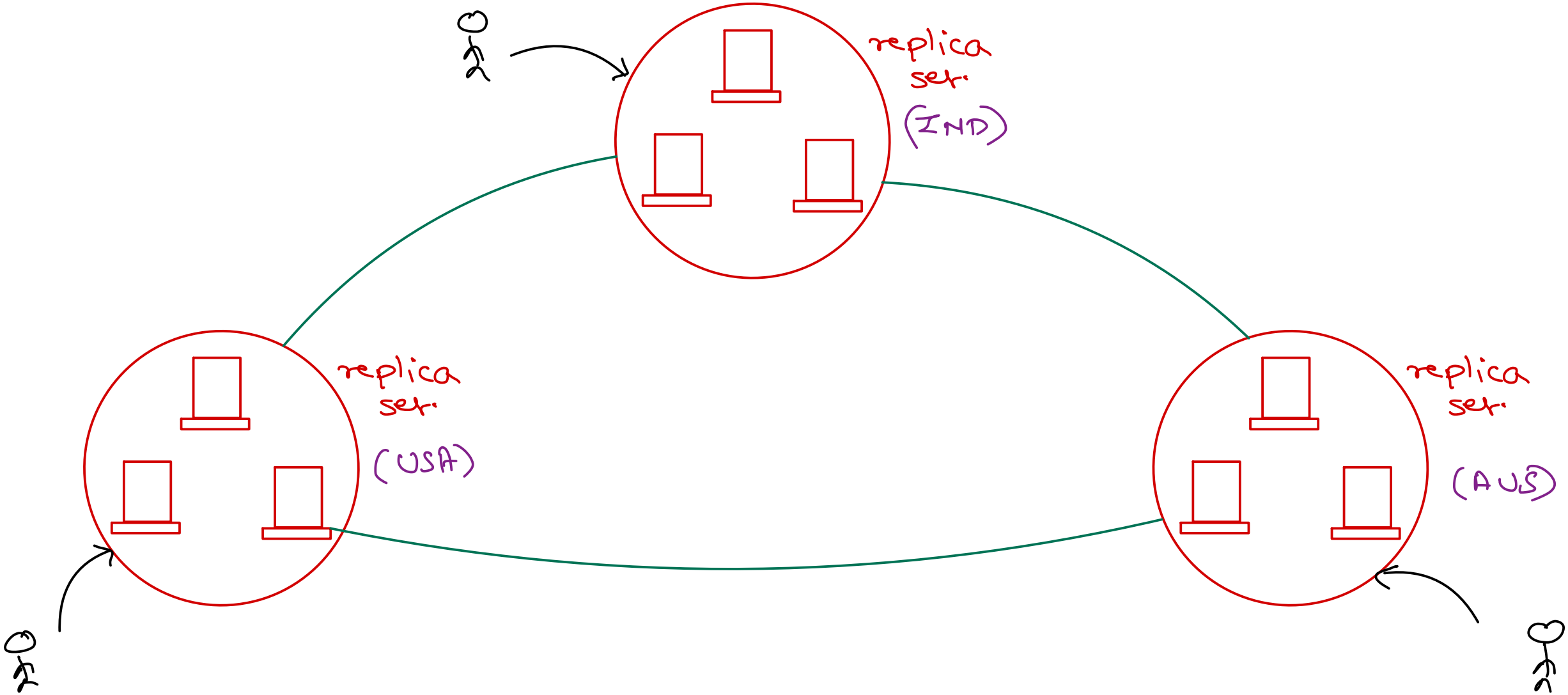
NoSQL Databases

Trainer: Mr. Nilesh Ghule

CAP (Brewer's) Theorem

- **Consistency** - Data is consistent after operation. After an update operation, all clients see the same data.
- **Availability** - System is always on (i.e. service guarantee), no downtime.
- **Partition Tolerance** - System continues to function even the communication among the servers is unreliable.
- Brewer's Theorem
  - It is impossible for a distributed data store to simultaneously provide more than two out of the above three guarantees.




replica set. (IND)
replica set. (USA)
replica set. (AUS)

Applications

- When to use NoSQL?
  - Large amount of data (TBs)
  - Many Read/Write ops
  - Economical Scaling
  - Flexible schema
- Examples:
  - Social media
  - Recordings
  - Geospatial analysis
  - Information processing

- When Not to use NoSQL?
  - Need ACID transactions
  - Fixed multiple relations
  - Need joins
  - Need high consistency
- Examples
  - Financial transactions
  - Business operations

RDBMS vs NoSQL

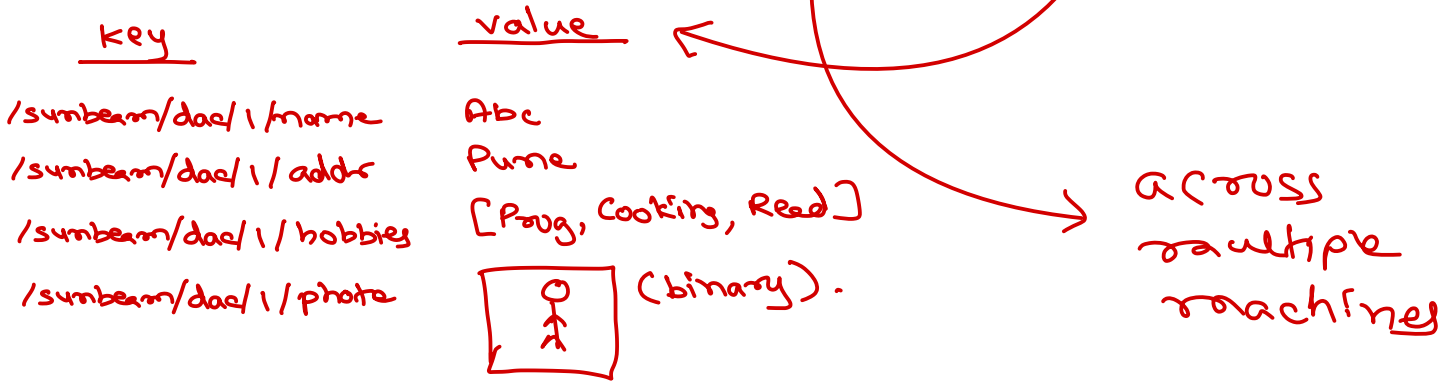
	RDBMS	NoSQL
Types	All types support SQL standard	Multiple types exists, such as document stores, key value stores, column databases, etc
History	Developed in 1970	Developed in 2000s
Examples	SQL Server, Oracle, MySQL	MongoDB, HBase, Cassandra, Redis, Neo4J
Data Storage Model	Data is stored in rows and columns in a table, where each column is of a specific type	The data model depends on the database type. It could be Key-value pairs, documents etc
Schemas	Fixed structure and schema	Dynamic schema. Structures can be accommodated
Scalability	Scale up approach is used (vertical)	Scale out approach is used (horizontal)
Transactions	Supports ACID and transactions	Supports partitioning and availability BASE
Consistency	Strong consistency	Dependent on the product [Eventual Consistency]
Support	High level of enterprise support	Open source model
Maturity	Have been around for a long time	Some of them are mature; others are evolving

NoSQL database

- NoSQL databases are non-relational.
- There is no standardization/rules of how NoSQL database to be designed.
- All available NoSQL databases can be broadly categorized as follows:
  - Key-value databases
  - Column-oriented databases
  - Graph databases
  - Document oriented databases

Key-value database

- Based on Amazon's Dynamo database.
- For handling huge data of any type.
- Keys are unique and values can be of any type i.e. JSON, BLOB, etc.
- Implemented as big distributed hash-table for fast searching.
- Example: redis, dynamodb, riak, ...

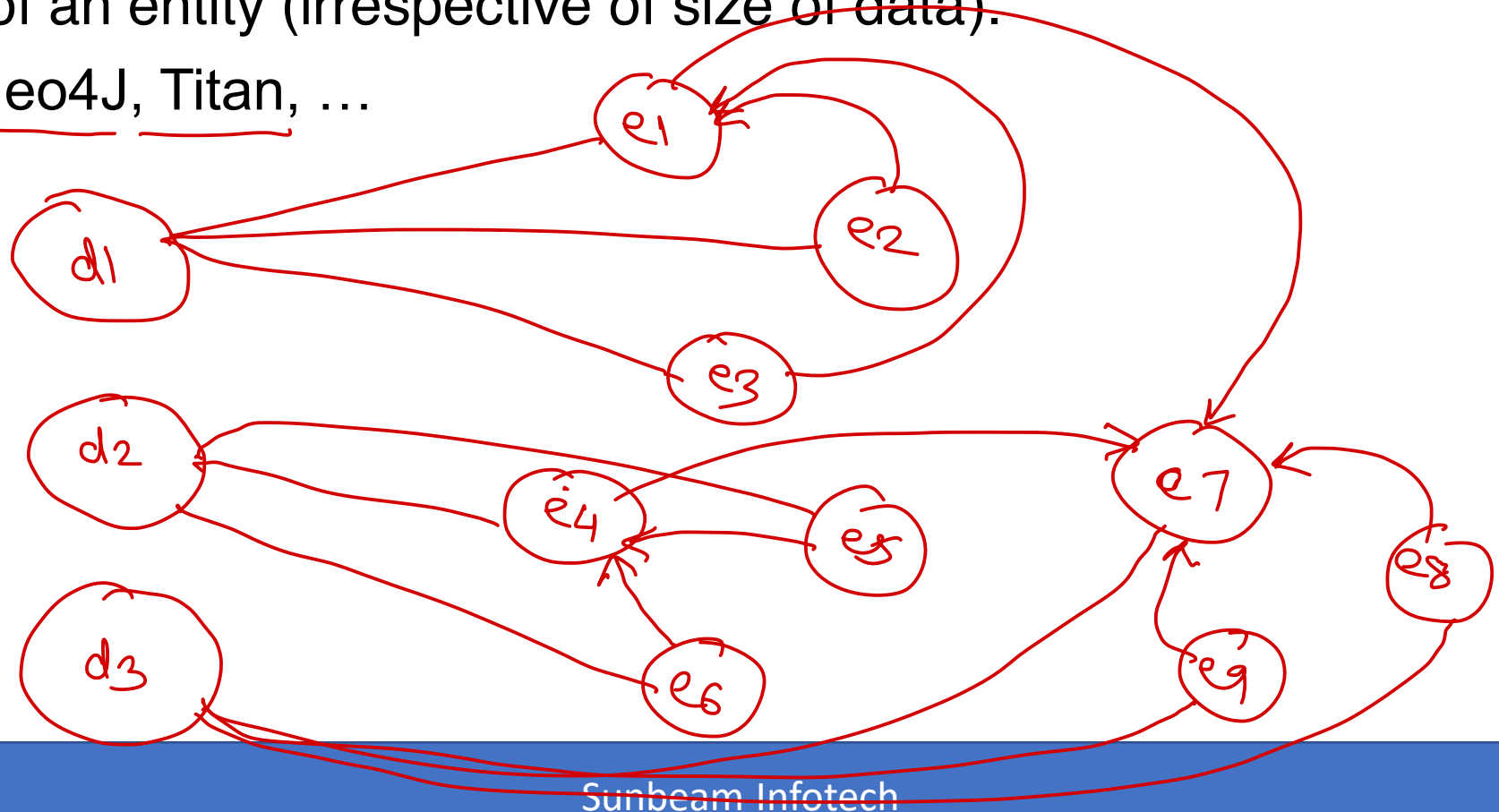


Column-oriented databases

- Values of columns are stored contiguously.
- Better performance while accessing few columns and aggregations.
- Good for data-warehousing, business intelligence, CRM, ...
- Examples: hbase, cassandra, bigtable, …

Graph databases

- Graph is collection of vertices and edges (lines connecting vertices).
- Vertices keep data, while edges represent relationships.
- Each node knows its adjacent nodes. Very good performance, when want to access all relations of an entity (irrespective of size of data).
- Examples: Neo4J, Titan, …



Document oriented databases

- Document contains data as key-value pair as JSON or XML.
- Document schema is flexible & are added in collection for processing.
- RDBMS tables → Collections
- RDBMS rows → Documents
- RDBMS columns → Key-value pairs in document
- Examples: MongoDb, CouchDb, …

b1 JSON → Java Script Object Notation.

```
{
  "id" : 1,          → int
  "title" : "Let us C",   → string
  "authar" : "kanetkar",
  "price" : 240.4    → double
}
```

r1 → JSON document

```
{ id : 1,
  name : "Nilesh",
  age : 38,
  hobbies : ["Program", "Reading", ...],
  addr : { area : "Katraj", city : "Pune", pin : 411046 },
  political : false,
  height : 5.9,
  bloodgroup : null
}
```

MongoDb Databases

Trainer: Mr. Nilesh Ghule

Agenda

- Introduction
- Installation
- JSON vs BSON
- Basic CRUD operations

Mongo Db

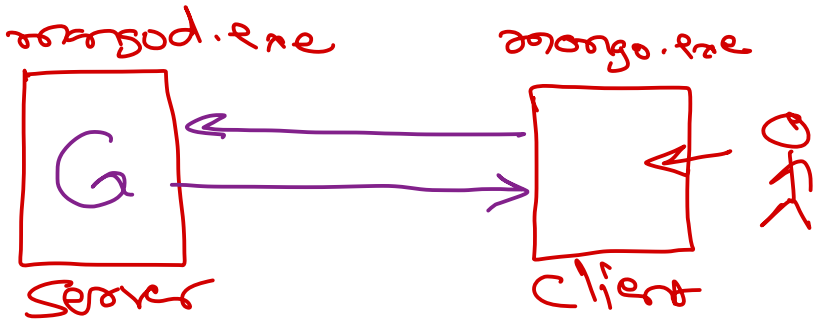
- Developed by 10gen in 2007
- Publicly available in 2009
- Open-source database which is controlled by 10gen
- Document oriented database → stores JSON documents
- Stores data in binary JSON. (BSON)
- Design Philosophy
  - MongoDB wasn't designed in a lab and is instead built from the experiences of building large scale, high availability, and robust systems.

JSON

- Java Script Object Notation
- Hierarchical way of organizing data
- Mongo stores JSON data into Binary form.

Mongo Server and Client

- MongoDb server (mongod) is developed in C, C++ and JS.
- MongoDb data is accessed via multiple client tools
  - mongo : client shell (JS).
  - mongofiles : stores larger files in GridFS.
  - mongoimport / mongoexport : tools for data import / export.
  - mongodump / mongorestore : tools for backup / restore.



- MongoDb data can be accessed in application through client drivers available for all major programming languages e.g. Java, Python, Ruby, PHP, Perl, …
- Mongo shell is follows JS syntax and allow to execute JS scripts.

MongoDb: Data Types

data	bson	values
null	10	
boolean	8	true, false
number	1 / 16 / 18	123, 456.78, NumberInt(“24”), NumberLong(“28”)
string	2	“....”
date	9	new Date(), ISODate(“yyyy-mm-ddThh:mm:ss”)
array	4	[…, …, …, …]
object	3	{ … }

Mongo - INSERT

- show databases;
- use database;
- db.contacts.insert({name: "nilesh", mobile: "9527331338"});
- db.contacts.insertMany([
  {name: "nilesh", mobile: "9527331338"},
  {name: "nitin", mobile: "9881208115"}
  ]);
- Maximum document size is 16 MB.
- For each object unique id is generated by client (if _id not provided).
  - 12 byte unique id :: [counter(3) | pid(2) | machine(3) | timestamp(4)]

field / key

client process id

client

Mongo – QUERY

- db.contacts.find(); → returns cursor on which following ops allowed:
  - hasNext(), next(), skip(n), limit(n), count(), toArray(), forEach(fn), pretty()
- Shell restrict to fetch 20 records at once. Press "it" for more records.
- db.contacts.find({ name: "nilesh" });
- db.contacts.find({ name: "nilesh" }, { _id:0, name:1 });
- Relational operators: $eq, $ne, $gt, $lt, $gte, $lte, $in, $nin
- Logical operators: $and, $or, $nor, $not
- Element operators: $exists, $type
- Evaluation operators: $regex, $where, $mod
- Array operators: $size, $elemMatch, $all, $slice

Mongo – DELETE

- db.contacts.remove(criteria);
- db.contacts.deleteOne(criteria);
- db.contacts.deleteMany(criteria);
- db.contacts.deleteMany({}); → delete all docs, but not collection
- db.contacts.drop(); → delete all docs & collection as well : efficient

Mongo – UPDATE

- db.contacts.update(criteria, newObj);
- Update operators: $set, $inc, $dec, $push, $each, $slice, $pull
- In place updates are faster (e.g. $inc, $dec, …) than setting new object. If new object size mismatch with older object, data files are fragmented.
- Update operators: $addToSet
- example: db.contacts.update({ name: "peter" },
- { $push : { mobile: { $each : ["111", "222"], $slice : -3 } } });
- db.contacts.update({ name: "t" }, { $set : { "phone" : "123" } }, true);
  - If doc with given criteria is absent, new one is created before update.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>