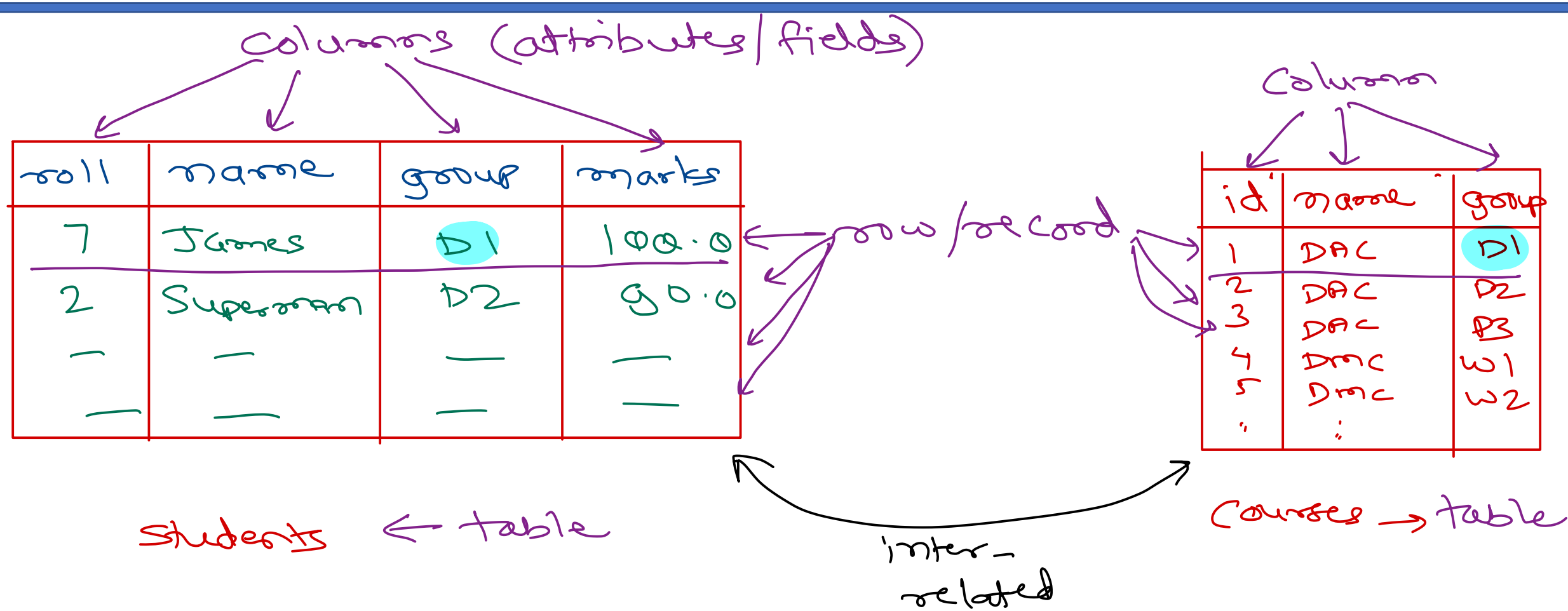
RDBMS & SQL Introduction

Trainer: Mr. Nilesh Ghule

DBMS

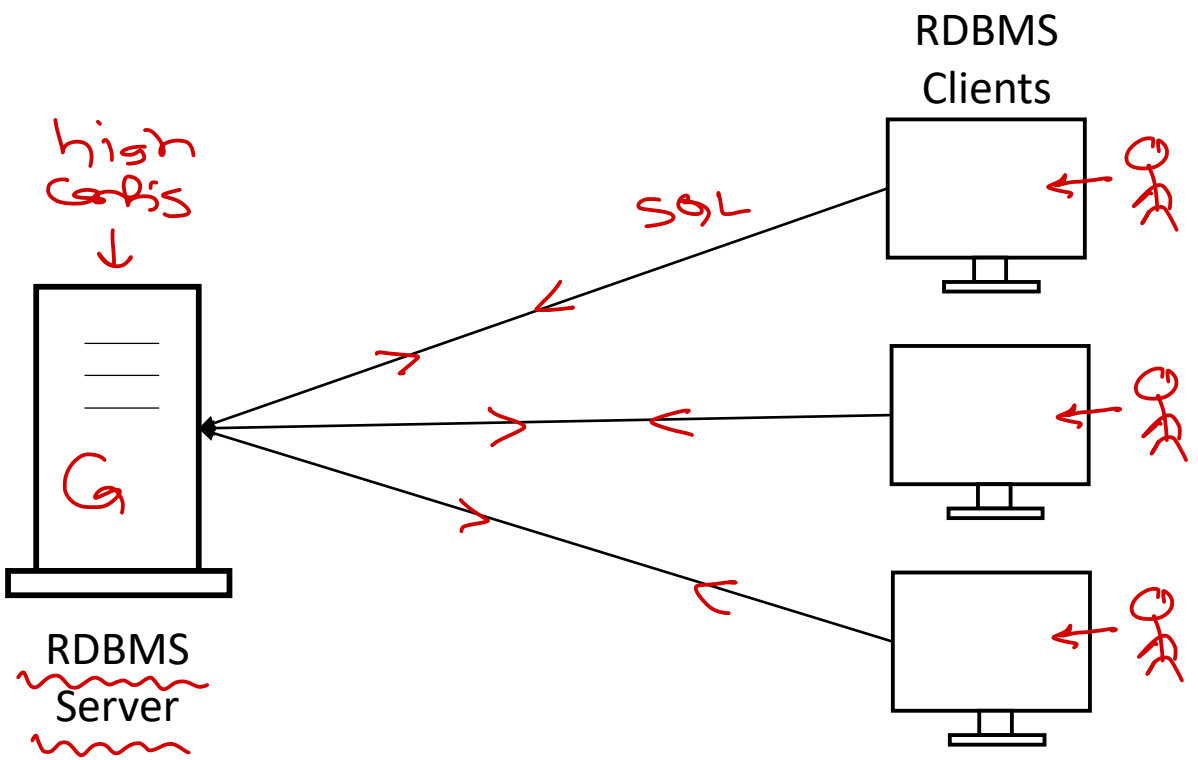
- Any enterprise application need to manage data.
- In early days of software development, programmers store data into files and does operation on it. However data is highly application specific.
- Even today many software manage their data in custom formats e.g. Tally, Address book, etc.
- As data management 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 table

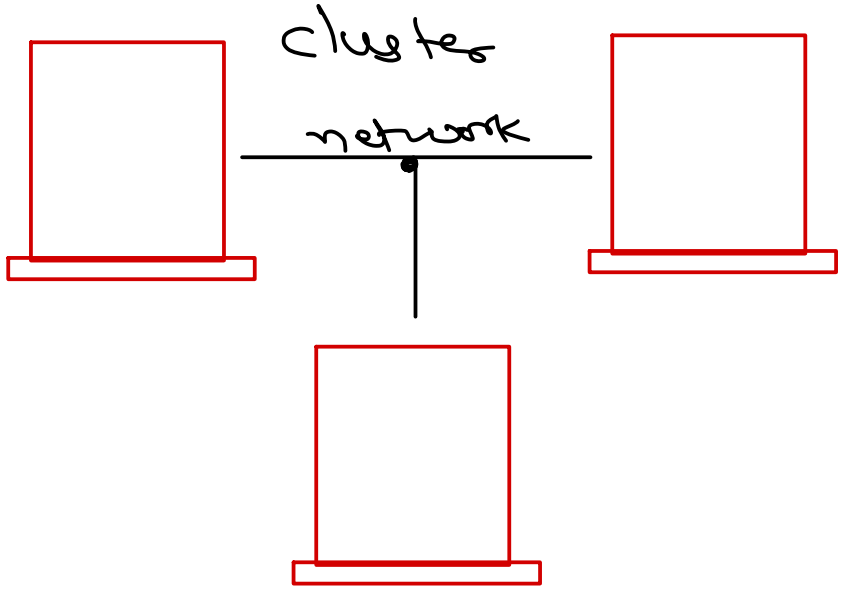

Columns (attributes/fields)
roll
name
group
marks
7
James
D1
100.0
2
Superman
D2
90.0
Students ← table
row/record
Column
id
name
group
1
DAC
D1
2
DAC
D2
3
DAC
D3
4
DMC
W1
5
DMC
W2
Courses → table
inter-related

RDBMS

- 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, …
- RDBMS design is based on Codd's rules developed at IBM (in 1970).



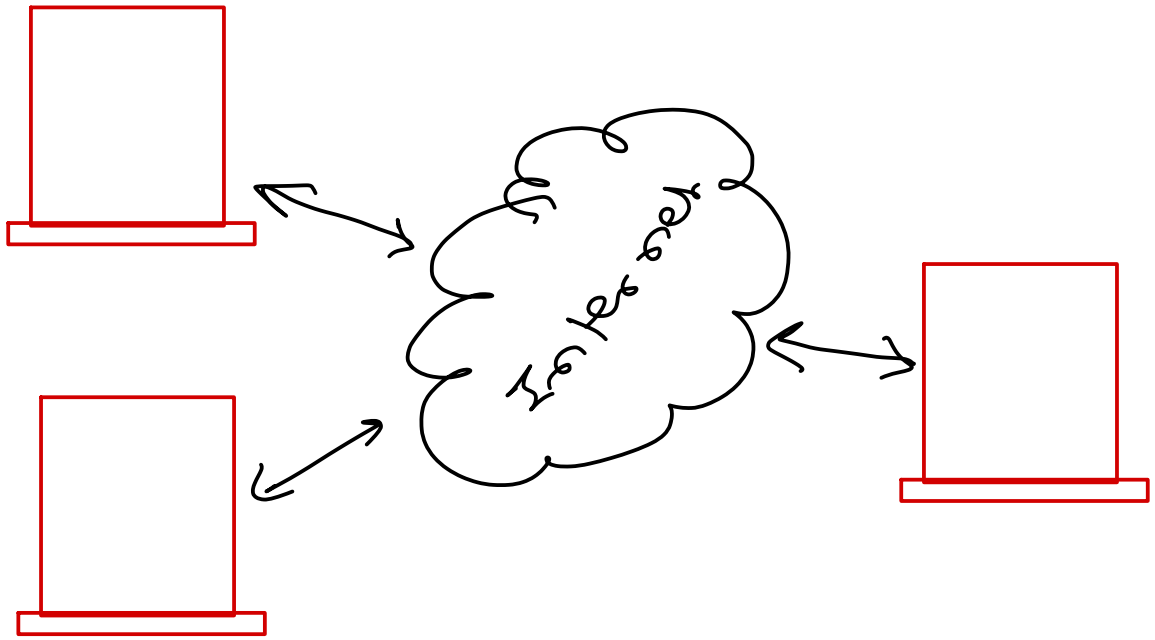
clustering - distributed systems.



eg. database cluster
webserver cluster.

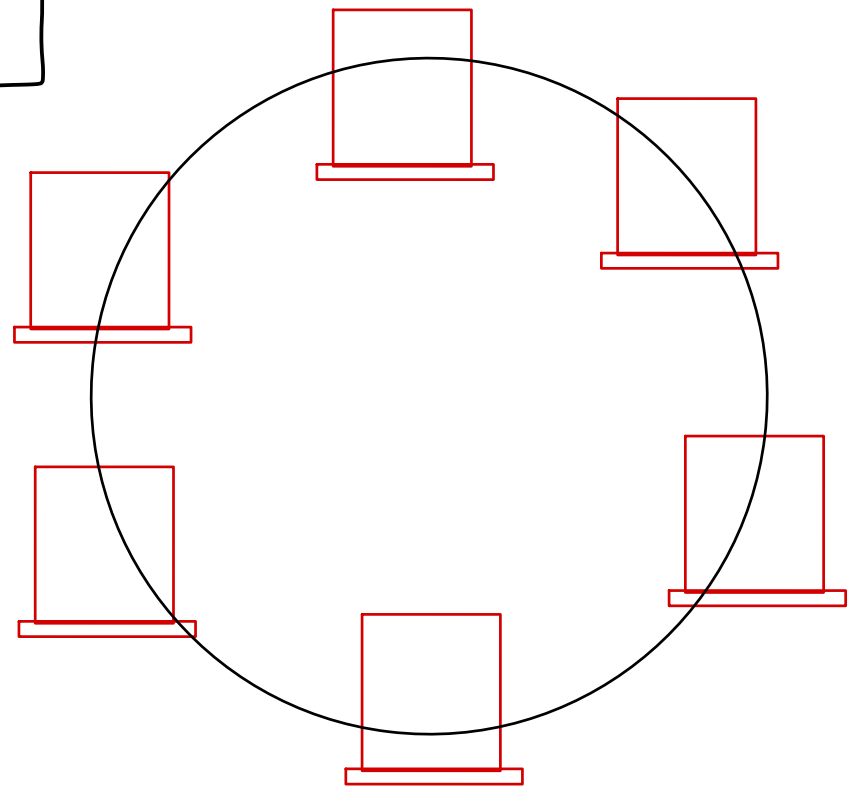
cluster is set of computers connected in a network for a specific task.

Internet



Internet

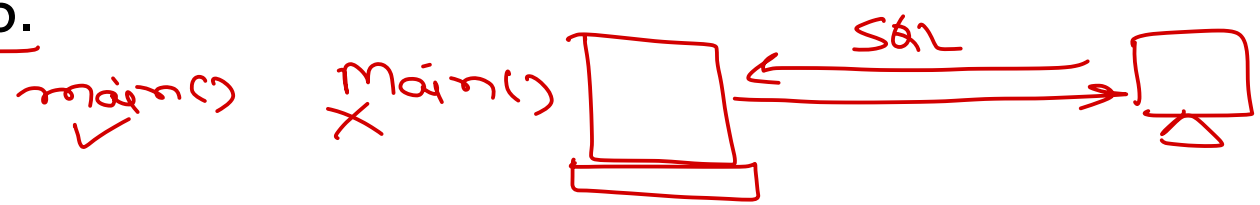
Star
Ring
Bus



LAN , MAN , WAN

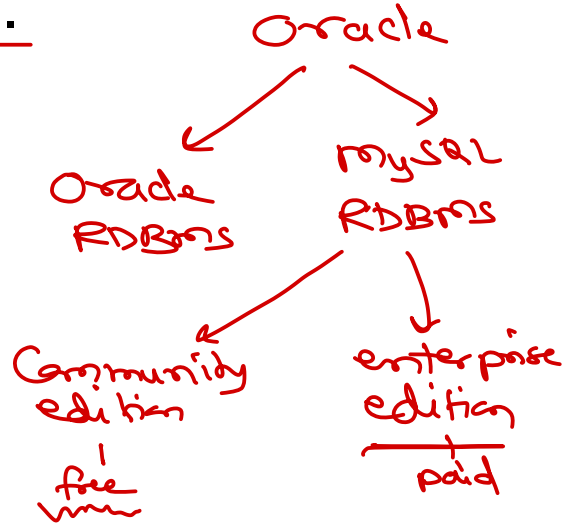
SQL - Structured Query Language.

- 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 - RDBMS Software → (of Oracle)

- 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

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

RDBMS & SQL Introduction

Trainer: Mr. Nilesh Ghule

developed in C/C++

mysql -u root -h localhost -p ← Password

user

host/server

mysql CLI

MySQL server

mysqld

Socket → ip + port

Common endpoint in a network

mysql

1) accept SQL query
2) process query
   - read data from disk
   - process data
   - create result
3) return result

port = 3306

port = 33060

mysql shell

mysql workbench

disk

mysql daemon

↓

no GUI ← run in background

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

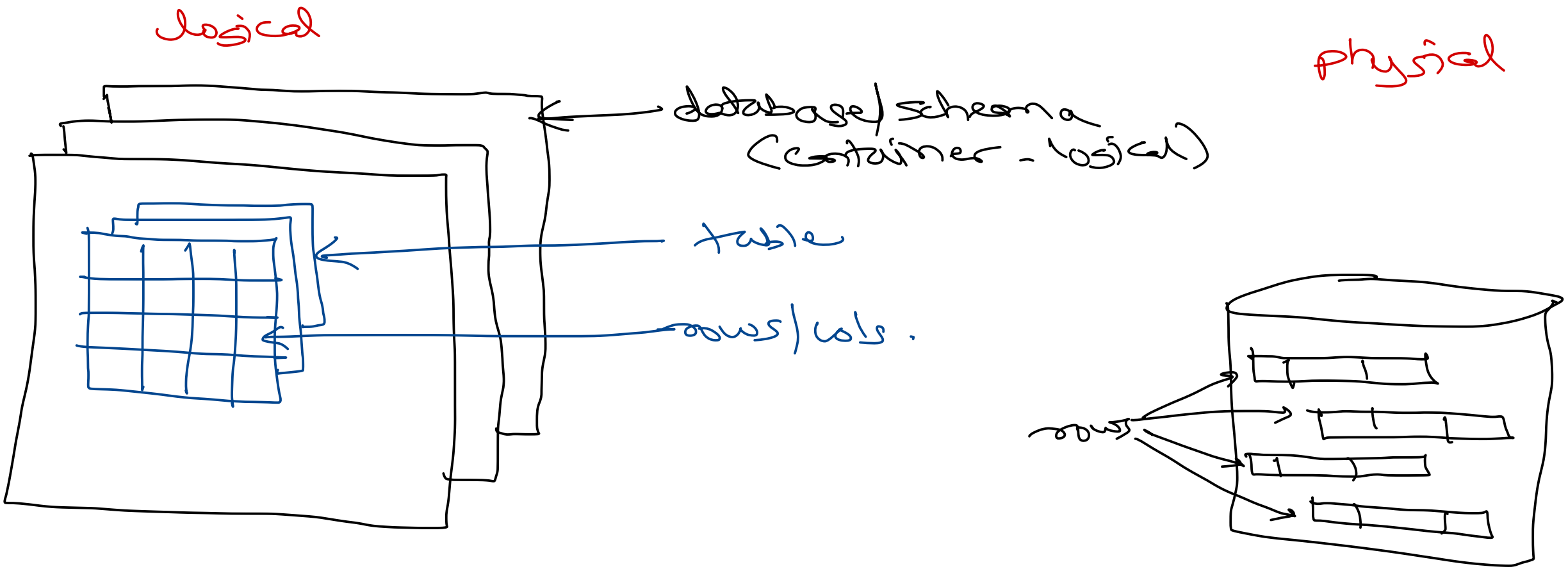
Getting started

- 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;

Database logical layout

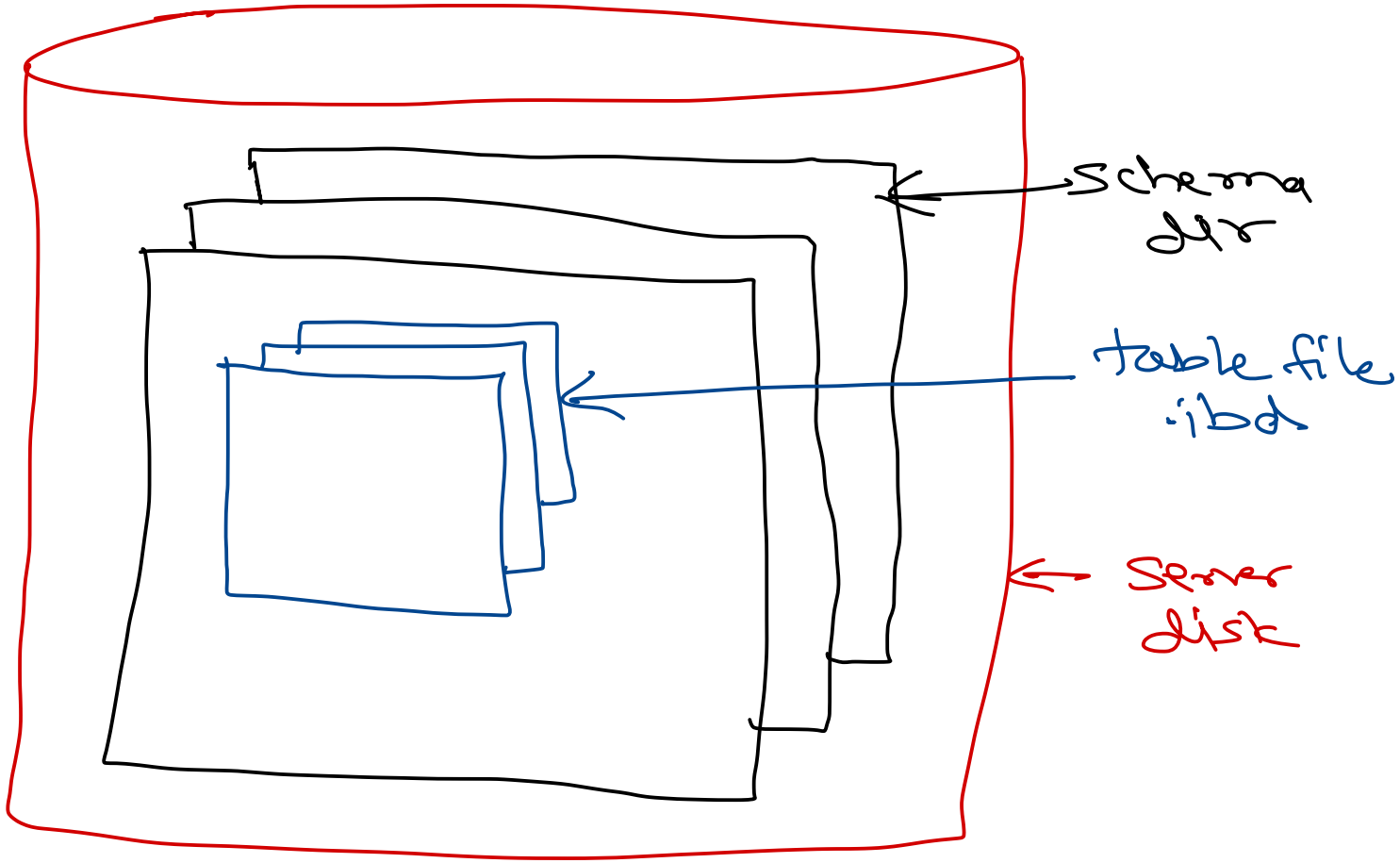
- Database/schema is like a namespace/container that stores all db objects related to a project.
- It contains tables, constraints, relations, stored procedures, functions, triggers, ...
- There are some system databases e.g. mysql, performance_schema, information_schema, sys, ... They contains db internal/system 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. → seperate lecture
- The data in table is in multiple rows.
- Each row have multiple values (as per columns).

fields


logical
physical
database/schema
(container - logical)
table
rows/data.
rows

Database physical layout

- In MySQL, the data is stored on disk in its data directory i.e. /var/lib/mysql
- 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.


Schema dir
table file .ibd
Server disk

Thank you!

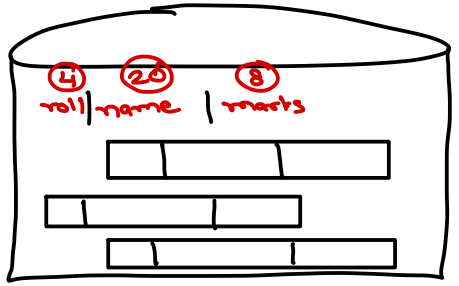
Nilesh Ghule <nilesh@sunbeaminfo.com>

RDBMS & SQL Introduction

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) I DECIMAL(m, n) – exact precision
  - Date/Time types
    - DATE, TIME, DATETIME, TIMESTAMP, YEAR(1)
  - 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



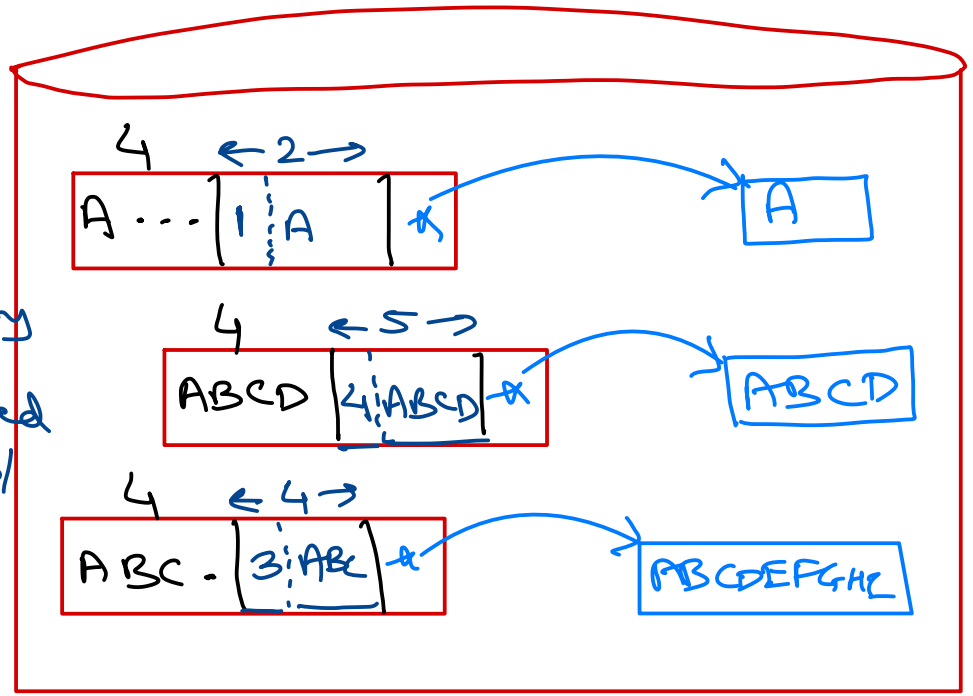
CHAR vs VARCHAR vs TEXT

String / date/time } data must be in '____';

- 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. → to record.
  - Very slow access.
  - Not ideal for indexing. → we will learn in indexes lecture.

char encoding

each char → how many bytes
→ how stored in mem/disk?



Server disk

- CREATE TABLE temp(c1 CHAR(4), c2 VARCHAR(4), c3 TEXT(4));
- DESC temp;
- INSERT INTO temp VALUES('abcd', 'abcd', 'abcdef');

Numerical data is not enclosed in '____'.

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.

each query

Thank you!

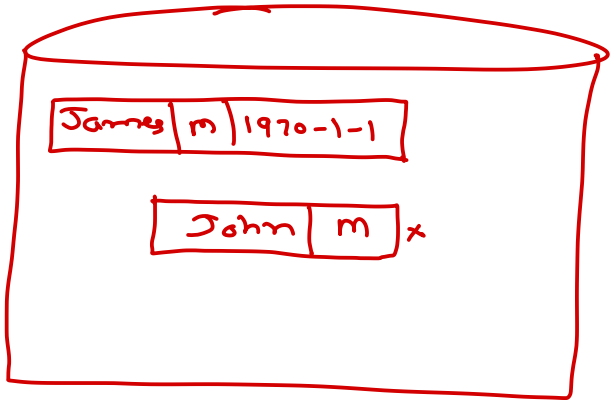
Nilesh Ghule <nilesh@sunbeaminfo.com>

RDBMS & SQL Introduction

Trainer: Mr. Nilesh Ghule

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;

SELECT – DQL

- 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;

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

RDBMS & SQL Introduction

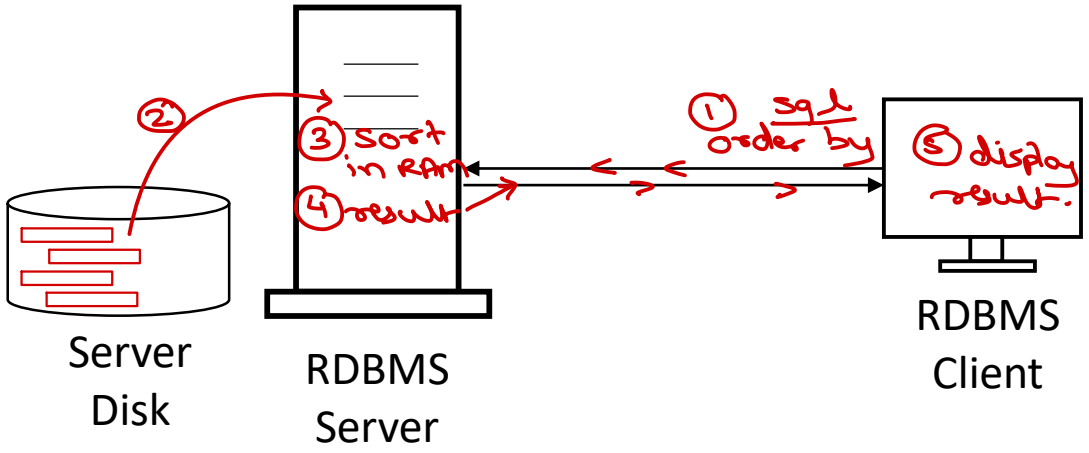
Trainer: Mr. Nilesh Ghule

SELECT – DQL

- Distinct values in column. → unique values.
  - 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;



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

operators
1) between
2) in
3) like
} monday lecture.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

RDBMS & SQL Introduction

Trainer: Mr. Nilesh Ghule

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.

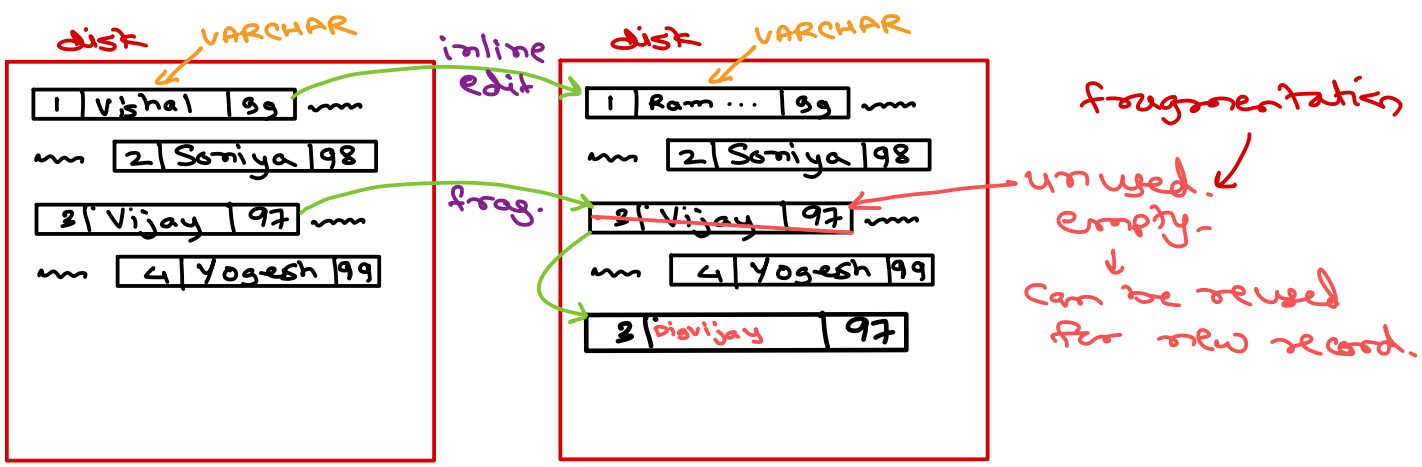
NOT BETWEEN

NOT IN

NOT LIKE

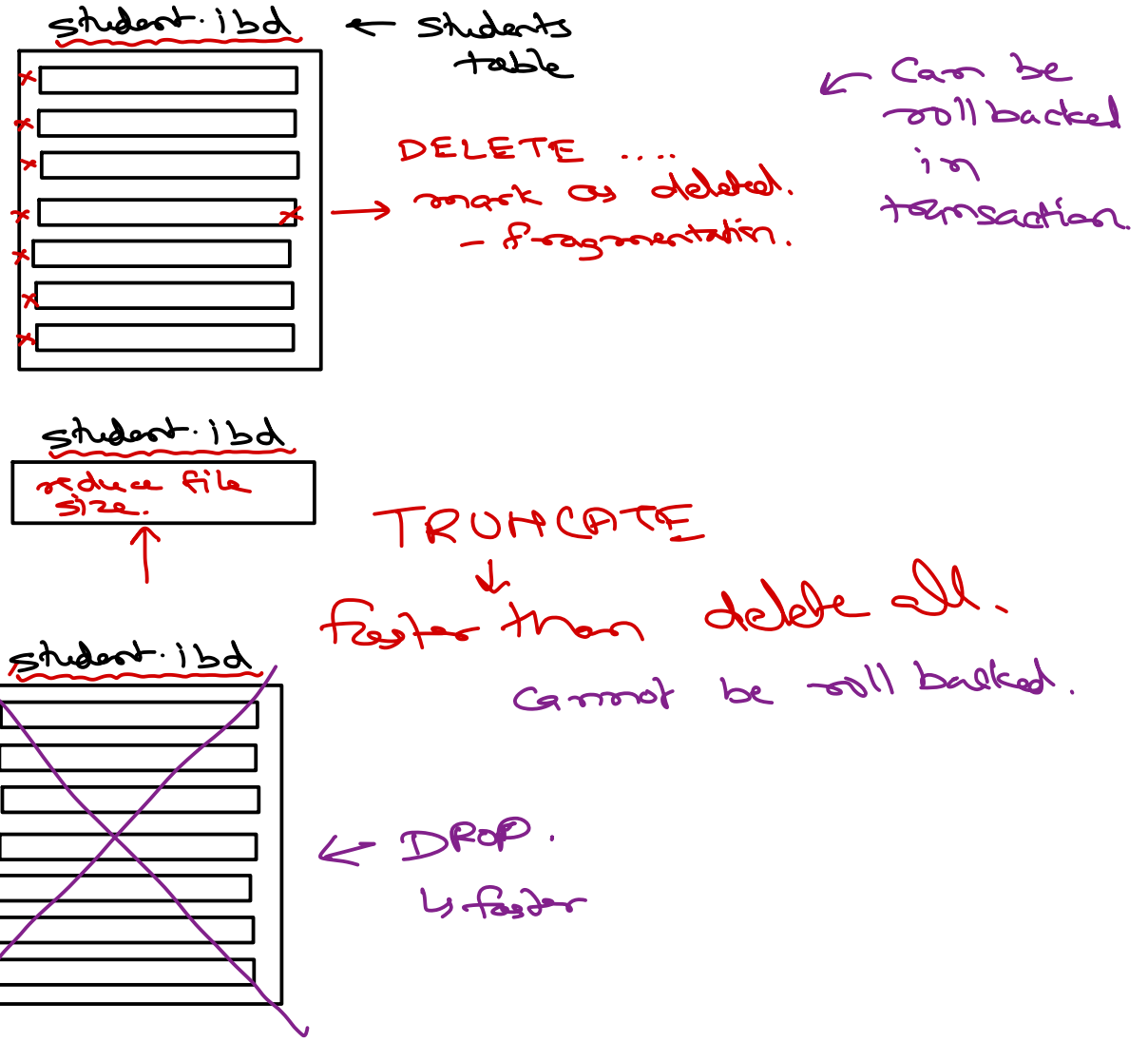
UPDATE – DML

- 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;



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Agneda

- WHERE clause
  - LIKE operator
  - IN operator
  - BETWEEN operator
  - ~~REGEXP operator~~
- DELETE, TRUNCATE & DROP
- UPDATE
- ~~DUAL table~~
- ~~SQL Functions~~

```
HELP SELECT;

SELECT columns FROM table_name
WHERE condition
ORDER BY column_expr
LIMIT n;
```

- String Size = Number of chars * Size of each char.
  - CHAR(10) = 10 * sizeof(char)
    - ASCII encoding --> 10 * 1 = 10 bytes
    - UTF-16 encoding --> 10 * 2 = 20 bytes

SELECT Query

WHERE clause

BETWEEN operator

- BETWEEN operator
  - WHERE column BETWEEN lower AND upper;
  - Range is inclusive of both ends.
- NOT BETWEEN -> reverse of BETWEEN

```
SELECT USER(), DATABASE();

-- find all emps having salary between 1500 to 3000 -- 1500 >= sal >= 3000.
SELECT * FROM emp WHERE sal >= 1500 AND sal <= 3000;

SELECT * FROM emp WHERE sal BETWEEN 1500 AND 3000;

-- find all emps hired in 1982.
SELECT * FROM emp WHERE hire >= '1982-01-01' AND hire <= '1982-12-31';

SELECT * FROM emp WHERE hire BETWEEN '1982-01-01' AND '1982-12-31';
```

```
-- find all emps having names between C and M.
SELECT * FROM emp WHERE ename BETWEEN 'C' AND 'M';
--| C      |
--| CLARK  |
--| FORD   |
--| JAMES  |
--| JONES  |
--| KING   |
--| M      |
--| MARTIN |
--| MILLER |
--| MZ     |
--| N      |

SELECT * FROM emp WHERE ename BETWEEN 'C' AND 'N';

SELECT * FROM emp WHERE ename BETWEEN 'C' AND 'N' AND ename != 'N';

SELECT * FROM emp WHERE ename BETWEEN 'C' AND 'MZ';

-- find all emps having salary not between 1500 to 3000.
SELECT * FROM emp WHERE sal NOT BETWEEN 1500 AND 3000;
```

IN operator

- IN operator check for "equality" with multiple values.
    - WHERE column IN (value1, value2, ...)
- NOT IN -- inverse of IN.

```
-- find all ANALYST, MANAGER and PRESIDENT.
SELECT * FROM emp WHERE job = 'ANALYST' OR job = 'MANAGER' OR job = 'PRESIDENT';

SELECT * FROM emp WHERE job IN ('ANALYST', 'MANAGER', 'PRESIDENT');

-- find all emps in dept 10 & 20.
SELECT * FROM emp WHERE deptno = 10 OR deptno = 20;

SELECT * FROM emp WHERE deptno IN (10, 20);

-- find emps JAMES, KING or MARTIN.
SELECT * FROM emp WHERE ename IN ('JAMES', 'KING', 'MARTIN');

-- find all emps not in dept 10 & 20.
SELECT * FROM emp WHERE deptno NOT IN (10, 20);

-- find all emps in dept 20 & 30 having sal in range 2000 to 2500.
SELECT * FROM emp
WHERE deptno IN (20, 30)
AND sal BETWEEN 2000 AND 2500;
```

LIKE operator

- In MySQL, String comparison case insensitive.
  - So 'JAMES' and 'James' are same.
- Comparing strings using wild-card chars (patterns).
  - % --> any number of any characters
  - _ --> single occurrence of any character

```
-- find all emps whose name start with 'S'.
SELECT * FROM emp WHERE ename LIKE 'S%';

-- find all emps whose name ends with 'D'.
SELECT * FROM emp WHERE ename LIKE '%D';

-- find all emps whose name contains 'IT'.
SELECT * FROM emp WHERE ename LIKE '%IT%';

-- find all emps whose name is of 4 letters.
SELECT * FROM emp WHERE ename LIKE '____';

-- find all emps whose second letter in 'A'.
SELECT * FROM emp WHERE ename LIKE '_A%';

-- find all emps whose name contains 'A' more than once.
SELECT * FROM emp WHERE ename LIKE '%A%A%';

-- find all emps whose name is in range 'T' to 'Z'.
SELECT * FROM emp
WHERE ename BETWEEN 'T' AND 'Z'
OR ename LIKE 'Z%';
-- BETWEEN T AND Z --> all names between T & Z, but not names Z onwards.
-- ename LIKE Z% --> all names starting with Z
-- OR combines both conditions.

-- find all emps whose name doesn't have I.
SELECT * FROM emp WHERE ename NOT LIKE '%I%';

-- find all emp with 4 letter name but start with A.
SELECT * FROM emp WHERE ename LIKE 'A___';
```

UPDATE query

- Update the row(s) and column(s) "values".
  - Cannot edit column name or type.
- UPDATE table_name SET column_name=new_value WHERE condition;

```
sunbeam> SELECT * FROM people;
+--------------------+--------+------------+
| name               | gender | birth      |
```

```
+--------------------+--------+------------+
| James Bond         | M      | 1960-02-23 |
| Superman           | M      | 1980-04-21 |
| Spiderman          | M      | 1984-12-01 |
| Batman             | M      | 1979-05-20 |
| John Galt          | M      | NULL       |
| Dagny Tagort       | F      | NULL       |
| Fransisco Dankonia | M      | NULL       |
+--------------------+--------+------------+
```

```
SELECT * FROM people;

UPDATE people SET birth='1960-01-01' WHERE name = 'John Galt';

SELECT * FROM people;

UPDATE people SET birth='1964-12-12' WHERE gender = 'F';

SELECT * FROM people;

UPDATE people SET gender='M', birth='1950-12-21';

SELECT * FROM people;
```

```
SELECT id, name, subject, price FROM books;

-- increase price of all books by 10%, if book name contains PROGRAMMING.
UPDATE books SET price = price + price * 0.10
WHERE name LIKE '%PROGRAMMING%';

-- decrease price of all C books by 5%.
 UPDATE books SET price = price - price * 0.05
 WHERE subject = 'C Programming';

-- set price of Herbert Schildt's book of Java to 1000.
UPDATE books SET price = 1000
WHERE author = 'Herbert Schildt' AND subject = 'Java Programming';
```

DELETE

- DELETE FROM table_name WHERE condition;

```
-- delete John Galt from people.
DELETE FROM people WHERE name = 'John Galt';

SELECT * FROM people;
```

```
-- delete all people whose name ends with 'man'
DELETE FROM people WHERE name LIKE '%man';

SELECT * FROM people;

-- delete all people
DELETE FROM people;

SELECT * FROM people;

DESCRIBE people;
```

- Q: DELETE * FROM tablename;
    - Error.
- Q: Why * is not used in DELETE?
    - "*" means all columns.
    - DELETE will delete the rows (not columns).
- Q. What happens in background when DELETE?

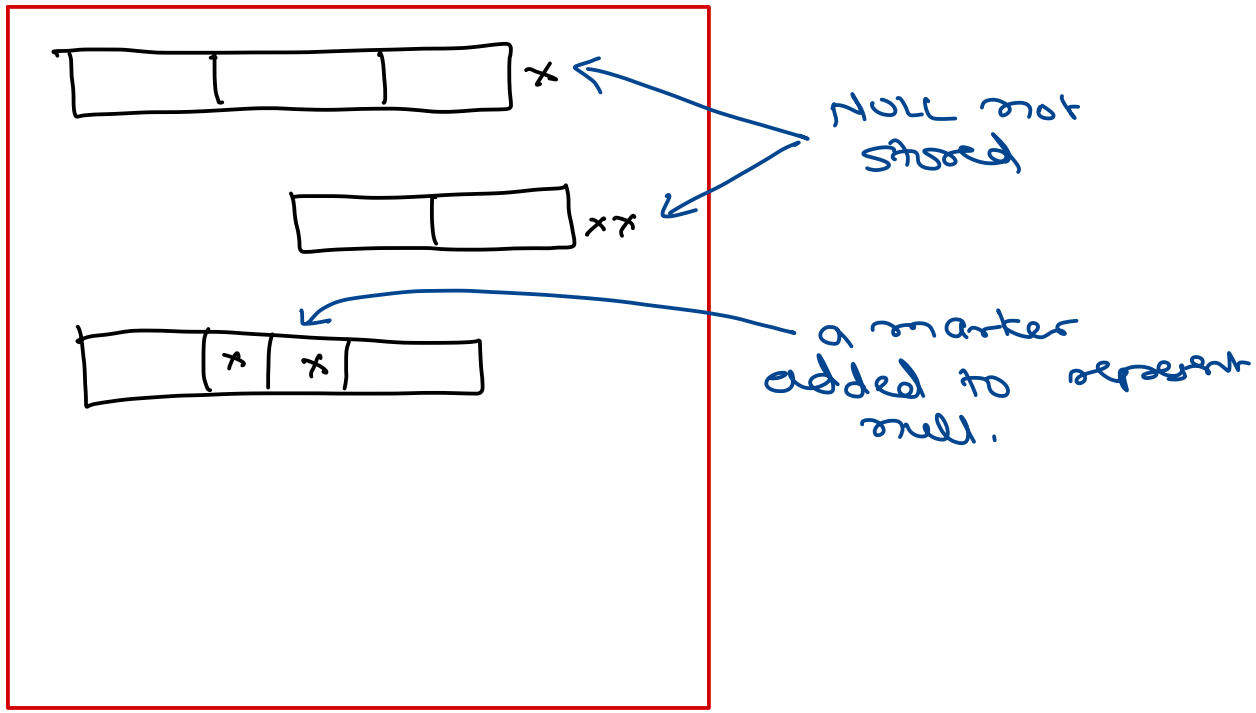
```
TRUNCATE TABLE students;

SELECT * FROM students;

DESCRIBE students;
```

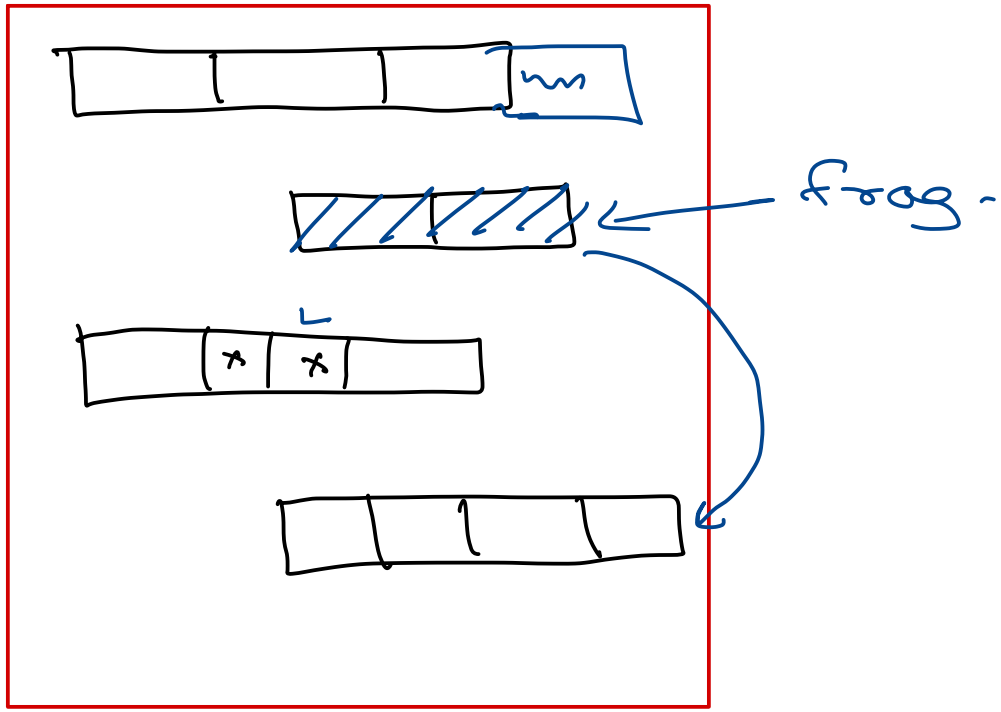
```
DROP TABLE students;
```

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

NULL





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

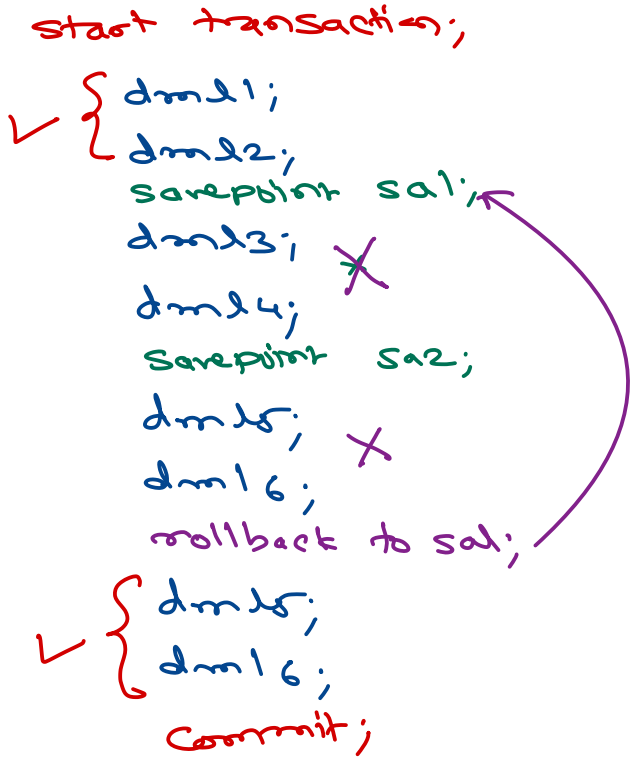
- Transaction management
  - START TRANSACTION;
  - …
  - COMMIT WORK;
  - START TRANSACTION;
  - …
  - ROLLBACK WORK;

WORK keyword optional.

- In MySQL autocommit variable is by default 1. So each DML command is auto-committed into database. → each query is one tx – auto commited.
  - 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".

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.
- 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.

Thank you!

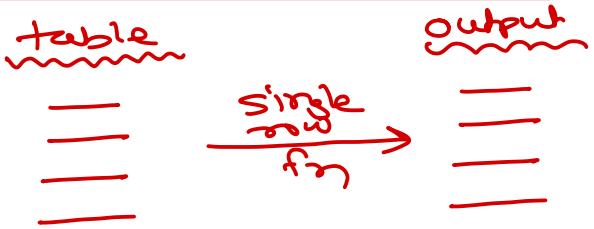
Nilesh Ghule <nilesh@sunbeaminfo.com>

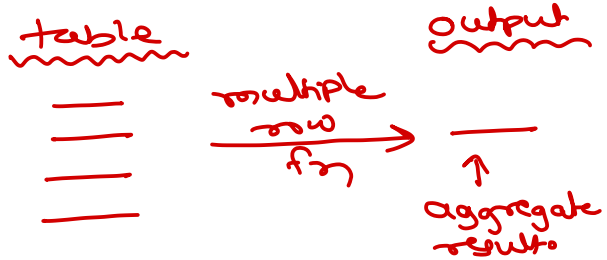
MySQL RDBMS

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
    - Values from multiple rows are aggregated to single value.
    - e.g. SUM(), MIN(), MAX(), …





- 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)

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

RDBMS - SQL

Agenda

- Transaction
- Row locking
- DUAL table
- SQL Functions

Transaction

- TCL - Transaction Control Language

```
SELECT USER(), DATABASE();

CREATE TABLE accounts(id INT, type CHAR(10), balance DECIMAL(10,2));

INSERT INTO accounts VALUES
(1, 'Saving', 20000),
(2, 'Saving', 500),
(3, 'Current', 200000),
(4, 'Saving', 40000);

SELECT * FROM accounts;

START TRANSACTION;

UPDATE accounts SET balance = balance - 1000 WHERE id = 1; -- dml1
UPDATE accounts SET balance = balance + 1000 WHERE id = 2; -- dml2

SELECT * FROM accounts;

COMMIT; -- finalize changes in rdbms (end of transaction)

SELECT * FROM accounts;

START TRANSACTION;

UPDATE accounts SET balance = balance - 1000 WHERE id = 1; -- dml1
UPDATE accounts SET balance = balance + 1000 WHERE id = 3; -- dml2

SELECT * FROM accounts;

ROLLBACK; -- discard changes in transaction  (end of transaction)

SELECT * FROM accounts;
```

```
START TRANSACTION;
INSERT INTO accounts VALUES(10, 'Saving', 10000);
INSERT INTO accounts VALUES(20, 'Saving', 10000);
SELECT * FROM accounts;
SAVEPOINT sa1;
INSERT INTO accounts VALUES(30, 'Saving', 10000);
INSERT INTO accounts VALUES(40, 'Saving', 10000);
SELECT * FROM accounts;
SAVEPOINT sa2;
INSERT INTO accounts VALUES(50, 'Saving', 10000);
INSERT INTO accounts VALUES(60, 'Saving', 10000);
SELECT * FROM accounts;
ROLLBACK TO sa2; -- discard 50 & 60 rows
SELECT * FROM accounts;
INSERT INTO accounts VALUES(70, 'Saving', 10000);
INSERT INTO accounts VALUES(80, 'Saving', 10000);
SELECT * FROM accounts;
COMMIT;
```

```
START TRANSACTION;

INSERT INTO accounts VALUES(90, 'Saving', 10000);
INSERT INTO accounts VALUES(100, 'Saving', 10000);

SELECT * FROM accounts;

DROP TABLE newpeople;
-- DDL query auto-commit current transaction.
SHOW TABLES;

ROLLBACK;
-- have no effect

SELECT * FROM accounts;
```

```
SELECT @@autocommit;

SET autocommit = 0;

SELECT @@autocommit;

INSERT INTO accounts VALUES (110, 'Saving', 10000);
INSERT INTO accounts VALUES (120, 'Saving', 10000);

ROLLBACK; -- or COMMIT

SELECT * FROM accounts;
```

```
DELETE FROM accounts;

SELECT * FROM accounts;

ROLLBACK; -- or COMMIT

SELECT * FROM accounts;

EXIT;
```

- Ensure than autocommit is 1.
- Open two different command prompts and login with two different users.

```
-- user1
SELECT * FROM accounts;

-- user2
SELECT * FROM accounts;

-- user1
START TRANSACTION;

-- user1
UPDATE accounts SET balance = balance - 1000 WHERE id = 1; -- dml1
UPDATE accounts SET balance = balance + 1000 WHERE id = 3; -- dml2
SELECT * FROM accounts;

-- user2
SELECT * FROM accounts;
-- changes from user1 are not visible

-- user1
COMMIT;

-- user2
SELECT * FROM accounts;
-- changes from user1 are visible
```

```
-- user1
SELECT * FROM accounts;

-- user2
SELECT * FROM accounts;

-- user1
START TRANSACTION;

-- user1
UPDATE accounts SET balance = balance - 1000 WHERE id = 1; -- dml1
```

```
UPDATE accounts SET balance = balance + 1000 WHERE id = 2; -- dml2
SELECT * FROM accounts;

-- user2
UPDATE accounts SET balance = balance - 1000 WHERE id = 3; -- dml1
-- query execution is blocked until timeout or user1 transaction is completed
(commit or rollback).
-- whole table was locked due to DML operation in the table (by user1)
UPDATE accounts SET balance = balance + 1000 WHERE id = 4; -- dml2
SELECT * FROM accounts;
```

- In MySQL by default whole table is locked when DML operations are performed in a transaction.
- To achieve row locking MySQL table must be indexed (or have primary key).

```
ALTER TABLE accounts ADD PRIMARY KEY(id);

-- user1
SELECT * FROM accounts;

-- user2
SELECT * FROM accounts;

-- user1
START TRANSACTION;

-- user1
UPDATE accounts SET balance = balance - 1000 WHERE id = 1; -- dml1
UPDATE accounts SET balance = balance + 1000 WHERE id = 2; -- dml2
SELECT * FROM accounts;

-- user2
UPDATE accounts SET balance = balance - 1000 WHERE id = 3; -- dml1
UPDATE accounts SET balance = balance + 1000 WHERE id = 4; -- dml2
SELECT * FROM accounts;

-- user2
UPDATE accounts SET balance = balance - 1000 WHERE id = 1; -- dml1
-- query execution is blocked until timeout or user1 transaction is completed
(commit or rollback).
-- only rows modified by user1 are locked (by user1)
```

```
-- user1
SELECT * FROM accounts;

-- user1
START TRANSACTION;

-- user1
SELECT * FROM accounts WHERE id = 1 FOR UPDATE;
```

```
-- lock account 1 for update/delete.

-- user2
SELECT * FROM accounts;

UPDATE accounts SET balance = balance - 1000 WHERE id = 1; -- dml1
-- will be blocked upto time or tx completion from user1

SELECT * FROM accounts WHERE id = 1 FOR UPDATE;
-- will be blocked upto time or tx completion from user1

-- user1
UPDATE accounts SET balance = balance - 1000 WHERE id = 1; -- dml1

COMMIT; -- or rollback
```

DUAL Table

- SELECT columns FROM tablename;
- DUAL table is a dummy/pseduo/virtual table for one row and one column.
- Used for executing arbitrary expression or few functions.
- You cannot perform DML & DDL operations on DUAL table.

```
SELECT USER();

SELECT USER() FROM DUAL;

SELECT 2 + 3 * 4;

SELECT 2 + 3 * 4 FROM DUAL;

SELECT NOW();

SELECT NOW() FROM DUAL;

DESCRIBE DUAL; -- error

SELECT * FROM DUAL; -- error
```

SQL functions

```
-- single row function
SELECT ename, LOWER(ename) FROM emp;

-- multi row function
SELECT SUM(sal) FROM emp;
```

String Functions

```
HELP Functions;

HELP String Functions;

HELP LOWER;

SELECT LOWER('Sunbeam Infotech');

SELECT UPPER('Sunbeam Infotech');

SELECT name, LOWER(name), UPPER(name) FROM books;

UPDATE books SET name = UPPER(name);

SELECT id, name FROM books;

SELECT CONCAT('Nilesh', ' ', 'Ghule');

SELECT ename, job, sal FROM emp;

SELECT CONCAT(ename, job), sal FROM emp;

SELECT CONCAT(ename, ' ', job), sal FROM emp;

SELECT CONCAT(ename, ' ', job) AS name_job, sal FROM emp;

SELECT CONCAT(empno, '-', ename), sal FROM emp;

SELECT ename, sal, comm, CONCAT(sal, '-', comm) FROM emp;

SELECT ASCII('A'), ASCII('a');
SELECT ASCII('ABCD');

SELECT CHAR(65);
SELECT CHAR(65 USING ASCII); -- A

SELECT UPPER(NULL), LOWER(NULL), ASCII(NULL);

SELECT TRIM('    ABCD     ');
SELECT LTRIM('    ABCD     ');
SELECT RTRIM('    ABCD     ');

SELECT LPAD('Sunbeam', 10, '*');
SELECT RPAD('Sunbeam', 10, '*');
SELECT RPAD( LPAD('Sunbeam', 17, '*'), 27, '*');

-- syntax: SUBSTRING(expression, start_pos, length)

-- substring start position +ve means from left
SELECT SUBSTRING('SUNBEAM', 4, 2);
```

```
SELECT SUBSTRING('SUNBEAM', 4, 4);
SELECT SUBSTRING('InfoTech', 3, 3);
SELECT SUBSTRING('SUNBEAM', 10, 3);

-- substring start position -ve means from right
SELECT SUBSTRING('SUNBEAM INFOTECH', -4, 4);
SELECT SUBSTRING('SUNBEAM INFOTECH', -8, 4);

-- in mysql, length cannot be -ve. Prints empty string.
SELECT SUBSTRING('SUNBEAM INFOTECH', 4, -2);

-- display first letter of all emp names.
SELECT SUBSTRING(ename, 1, 1) FROM emp;

-- find all emps starting with 'M'.
SELECT ename FROM emp WHERE SUBSTRING(ename, 1, 1) = 'M';

-- find all emps whose name start from 'C' to 'M'.
SELECT ename FROM emp WHERE SUBSTRING(ename, 1, 1) BETWEEN 'C' AND 'M';

SELECT LEFT('Sunbeam Infotech', 4);
SELECT RIGHT('Sunbeam Infotech', 4);
```

Numeric Functions

```
HELP Numeric Functions;

SELECT POWER(2, 5);
SELECT POWER(2, 0.5);
SELECT SQRT(2);

SELECT ABS(-3), ABS(6);

SELECT ROUND(1234.5678, 2), ROUND(1234.4321, 2);
SELECT ROUND(1234.5678, -2), ROUND(3456.1234, -2);
SELECT ROUND(1234.5678, 0), ROUND(3456.1234, 0);

SELECT ROUND(5678.12, -4);

-- CEIL() --> Round-Up to next "Int".
SELECT CEIL(3.4), CEIL(-3.4);

-- FLOOR() --> Round-Down to prev "Int".
SELECT FLOOR(3.4), FLOOR(-3.4);
```

Date and Time Functions

- Data Types
  - DATE --> calendar DATE --> yyyy-mm-dd
    - '1000-01-01' to '9999-12-31'

- TIME --> time duration --> hh:mm:ss
  - -838:59:59 to 838:59:59
- DATETIME --> calendar date and wall time --> yyyy-mm-dd hh:mm:ss
  - '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
- TIMESTAMP --> number of seconds from epoch time '01-01-1970 00:00:00' --> stored as int
  - '1970-01-01 00:00:00' to '2038-01-19 03:14:07'
- YEAR --> stored as int (1)
  - 1901 to 2155

```
SELECT NOW(), SYSDATE();

SELECT NOW(), SLEEP(5), SYSDATE();

SELECT DATE(NOW()), TIME(NOW());

SELECT DAYOFMONTH(NOW());
SELECT MONTH(NOW());
SELECT MONTHNAME(NOW());
SELECT YEAR(NOW());
SELECT WEEKDAY(NOW());

-- find all emps hired on Tuesday
SELECT * FROM emp WHERE WEEKDAY(hire) = 1;

-- arg1 > arg2 --> +ve value (number of days)
SELECT DATEDIFF(NOW(), '1983-09-28');

SELECT TIMESTAMPDIFF(YEAR, '1983-09-28', NOW());
SELECT TIMESTAMPDIFF(MONTH, '1983-09-28', NOW());
SELECT TIMESTAMPDIFF(DAY, '1983-09-28', NOW());
SELECT TIMESTAMPDIFF(SECOND, '1983-09-28', NOW());

SELECT DATE_ADD(NOW(), INTERVAL 28 DAY);

-- print experience of all emps in months.
SELECT ename, hire, TIMESTAMPDIFF(MONTH, hire, NOW()) exp FROM emp;
```

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

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()

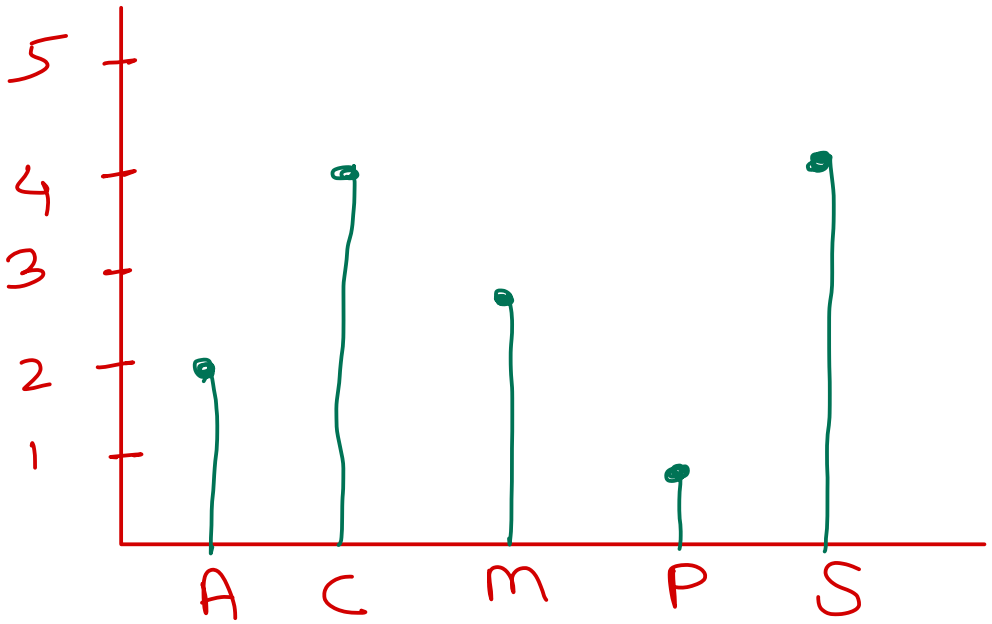
- IF(condition, true-value, false-value)

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.

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.


5
4
3
2
1
A
C
M
P
S

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL RDBMS

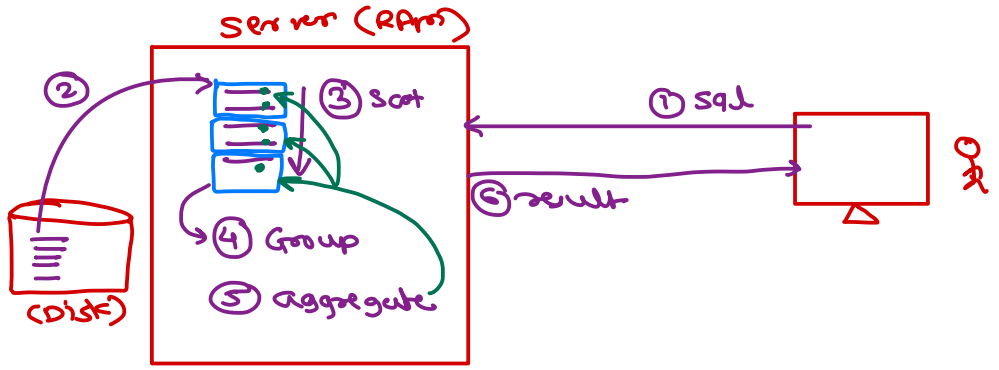
Trainer: Mr. Nilesh Ghule

GROUP BY clause

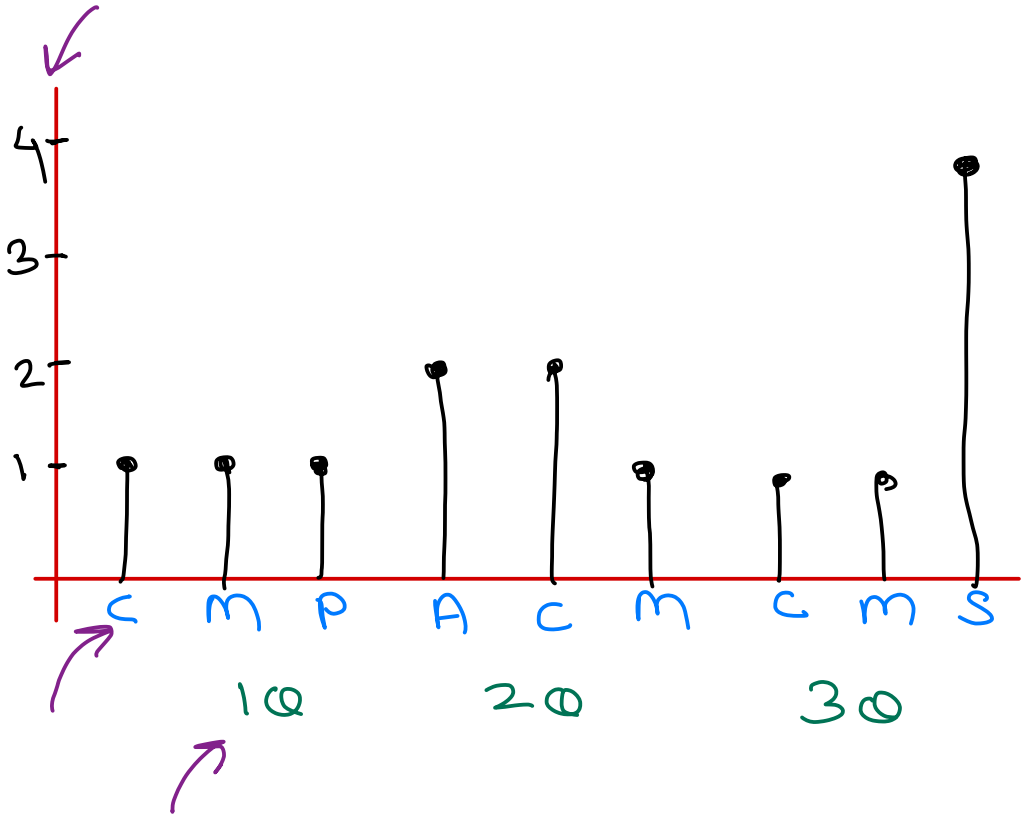
- GROUP BY is used for analysis of data i.e. generating reports & charts.
- When GROUP BY single column, generated output can be used to plot 2-D chart. When GROUP BY two column, generated output can be used to plot 3-D chart and so on.
- GROUP BY queries are also called as Multi-dimensional / Spatial 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.

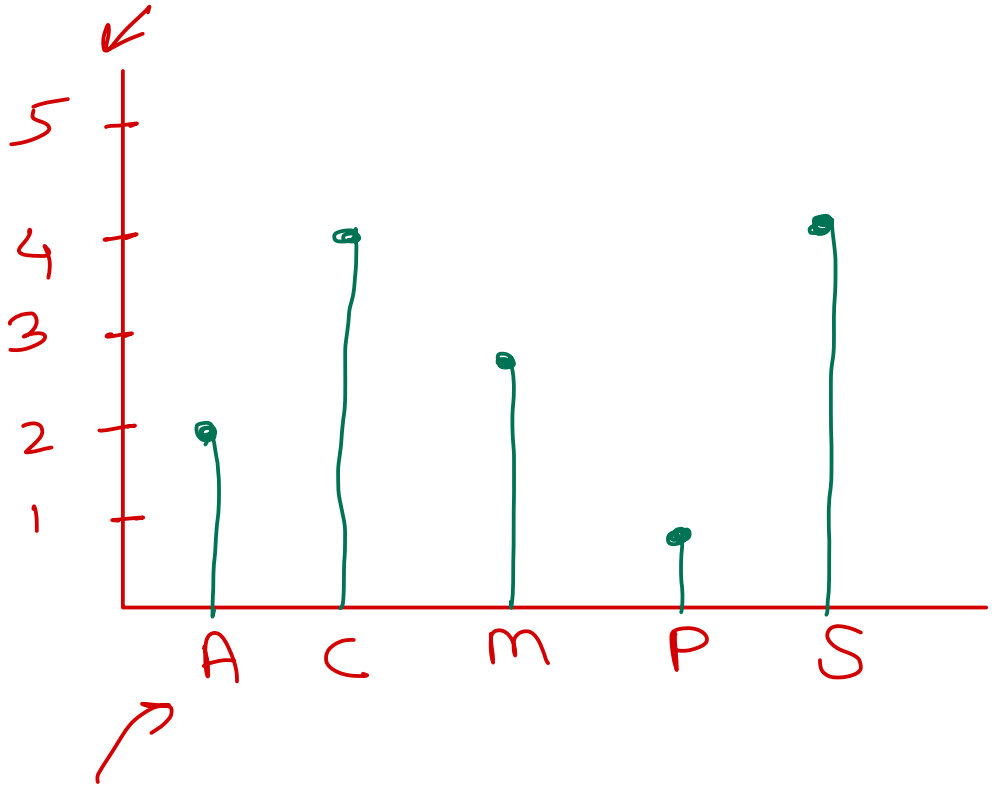
```
select job, count(empno) from emp group by job
having job in (M, A);
```

- 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 job, count(empno) from emp where job in (M, A)
group by job;
```


4
3
2
1
C
M
P
A
C
M
C
M
S
100
200
300


5
4
3
2
1
A
C
M
P
S

HAVING clause

- HAVING clause cannot be used without GROUP BY clause.
- HAVING clause is used to specify condition on aggregate values/fns.
- Examples:
  - SELECT deptno, SUM(sal) FROM EMP GROUP BY deptno HAVING SUM(sal) > 9000;
- Syntactical Characteristics:
  - WHERE clause executed for each record; while HAVING is executed for each group.
  - HAVING clause can be used to specify condition on group fn or grouped columns.
  - However using HAVING to specify condition of group col reduce the performance. Use WHERE clause for the same.
- Examples:
  - SELECT deptno, SUM(sal) FROM EMP GROUP BY deptno HAVING deptno = 20;
  - SELECT deptno, SUM(sal) FROM EMP WHERE deptno = 20 GROUP BY deptno; ← efficient.
- We may use GROUP BY with WHERE, ORDER BY & LIMIT.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Agenda

- Transaction internals
- SQL Functions
    - NULL values
- Group By clause
- Having clause
- REGEXP operator
- ~~Table Relations~~

Q & A

```
SELECT ename, hire, DATE_FORMAT(hire, '%W') FROM emp;

SELECT ROUND(5678.12, -4);
-- round upto 4 places before decimal --> 10000
-- 5678.12 >= 5000 --> 10000

SELECT ROUND(23493.12, -4);
-- 23493.12 < 2500 --> 20000

SELECT ename, hire, LAST_DAY(hire) FROM emp;

SELECT ename, REPLACE(ename, 'IT', 'XX') FROM emp;

SELECT ename, REPLACE(REPLACE(ename, 'I', 'X'), 'T', 'Y') FROM emp;

SELECT ename FROM emp WHERE LENGTH(ename) = 4;

-- SUBSTRING(string, start_pos, length);
-- start_pos is 1 based (not 0 based).
--      +ve value means from the start, and -ve value means from the end.
-- length is num of chars.
--      only +ve value allowed.
--      <=0 gives empty string
```

Transaction internals

SQL Functions

NULL related

- Most of SQL functions result NULL when NULL is passed as one of its arg.

```
SELECT SUBSTRING('Sunbeam', NULL, 4);

SELECT POWER(2, NULL);

SELECT DATE_FORMAT('2021-05-21', NULL);

-- print total income of emp.
SELECT ename, sal, comm, sal + comm AS income from emp;
```

- Few functions are designed to be used with NULL.
  - IFNULL(expr, value) --> if expr is NULL, consider this value.
    - IFNULL(comm, 0.0) --> if comm is NULL, consider it as 0.
  - NULLIF(expr, value) --> if expr is equal to given value, then consider result as NULL.
  - ISNULL(expr) --> returns 1 if null, else 0.
  - COALESCE(expr1, expr2, expr3) --> returns first non-null value.

```
SELECT ename, sal, comm, IFNULL(comm, 0.0) FROM emp;

SELECT ename, sal, comm, sal + IFNULL(comm, 0.0) AS income FROM emp;

SELECT ename, sal, NULLIF(sal, 3000) FROM emp;

SELECT ename, comm, NULLIF(comm, 0.0) FROM emp;

-- find emps who do not get comm (or comm is 0).
SELECT ename, comm FROM emp WHERE NOT (comm IS NULL OR comm = 0.0);

SELECT ename, comm FROM emp WHERE NULLIF(comm, 0.0) IS NOT NULL;

-- ISNULL function is similar to IS NULL operator.
SELECT ename, comm, ISNULL(comm) FROM emp;

SELECT ename, comm FROM emp WHERE ISNULL(comm) = 1;

SELECT ename, comm FROM emp WHERE comm IS NULL;

-- COALESCE
SELECT COALESCE(NULL, NULL, 1, 'A', '2021-02-12');

SELECT COALESCE(NULL, NULL, NULL, 'A', '2021-02-12');

-- if comm is null, return sal; otherwise return comm
SELECT ename, sal, comm, COALESCE(comm, sal) FROM emp;
```

List Functions (Single Row Fn)

- CONCAT(expr1, expr2, ...)
- COALESCE(expr1, expr2, ...)

- GREATEST(expr1, expr2, ...) --> return max from the list
- LEAST(expr1, expr2, ...) --> return min from the list

```
SELECT sal, comm, GREATEST(sal, IFNULL(comm,0.0)) AS gr FROM emp;

SELECT sal, comm, LEAST(sal, IFNULL(comm,0.0)) AS gr FROM emp;
```

Control Function

- IF(condition, true-expr, false-expr) --> similar to ternary operator

```
SELECT ename, sal, IF(sal >= 2500, 'Rich', 'Poor') FROM emp;
```

Multi-Row/Group Functions

- Aggregate operations
  - SUM(), AVG(), COUNT(), MAX(), MIN(), STDEV(), ...
- Aggregate operation on multiple values from multiple rows.
- SUM(sal) --> returns sum of sal of all emps (14 rows --> 14 sals)

```
SELECT SUM(sal) FROM emp;

SELECT SUM(sal), AVG(sal), MAX(sal), MIN(sal), COUNT(sal) FROM emp;

SELECT SUM(comm), AVG(comm), MAX(comm), MIN(comm), COUNT(comm) FROM emp;
```

- Limitations of Group Functions

```
SELECT @@sql_mode;

-- temp change (will be discarded when mysql client is closed)
SET sql_mode = 'ONLY_FULL_GROUP_BY';

SELECT @@sql_mode;

-- cannot select column with group function
SELECT ename, SUM(sal) FROM emp;

-- cannot select single row function with group function
SELECT LOWER(ename), SUM(sal) FROM emp;

-- find emp with max sal.
-- cannot use group function in WHERE clause.
SELECT * FROM emp WHERE sal = MAX(sal);
```

```
-- cannot call group function in another group function.
SELECT MAX(SUM(sal)) FROM emp;
```

- To make SQL mode permanent.
  - In C:\ProgramData\MySQL\MySQL Server 8.0\my.ini
    - Under [mysqld]
      - sql-mode="STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION,ONLY_FULL_GROUP_BY"
  - Restart the computer
  - Login to MySQL CLI.
    - SELECT @@sql_mode;

GROUP BY

- Allows me to SELECT grouped-column with GROUP functions

```
SELECT * FROM emp;

-- on which dept company is spending how much on salaries?
SELECT deptno, SUM(sal) FROM emp; -- error
SELECT deptno, SUM(sal) FROM emp GROUP BY deptno;

-- in which dept how many emps are there?
SELECT deptno, COUNT(empno) FROM emp GROUP BY deptno;

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

-- for each job how many emps are available?
SELECT job, COUNT(empno) FROM emp GROUP BY job;

-- how emps are hired in each year.
SELECT * FROM emp WHERE YEAR(hire) = 1980;
SELECT COUNT(empno) FROM emp WHERE YEAR(hire) = 1980;
SELECT COUNT(empno) FROM emp WHERE YEAR(hire) = 1981;
SELECT COUNT(empno) FROM emp WHERE YEAR(hire) = 1982;
SELECT COUNT(empno) FROM emp WHERE YEAR(hire) = 1983;

SELECT YEAR(hire), COUNT(empno) FROM emp
GROUP BY YEAR(hire);

-- get max sal per job.
SELECT job, MAX(sal) FROM emp
GROUP BY job;

-- count num of emps per dept, per job.
SELECT DISTINCT deptno FROM emp;

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

```
SELECT DISTINCT job FROM emp;

SELECT job, COUNT(empno) FROM emp GROUP BY job;

SELECT DISTINCT deptno, job FROM emp;

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

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

```
SELECT deptno, SUM(sal) FROM emp GROUP BY deptno;

SELECT SUM(sal) FROM emp GROUP BY deptno;

SELECT deptno, SUM(sal) FROM emp; -- error

SELECT deptno, SUM(sal) FROM emp GROUP BY deptno;
```

```
-- find the dept which spend max on sal of emps.
SELECT deptno, SUM(sal) FROM emp
GROUP BY deptno
ORDER BY SUM(sal) DESC;

SELECT deptno, SUM(sal) FROM emp
GROUP BY deptno
ORDER BY SUM(sal) DESC
LIMIT 1;

-- find the dept which spend max on sal of emps other managers.
SELECT * FROM emp WHERE job <> 'MANAGER';

SELECT SUM(sal) FROM emp WHERE job <> 'MANAGER'; -- total sal of emps other than
managers

SELECT deptno, SUM(sal) FROM emp WHERE job <> 'MANAGER'
GROUP BY deptno;

SELECT deptno, SUM(sal) FROM emp WHERE job <> 'MANAGER'
GROUP BY deptno
ORDER BY SUM(sal) DESC;

SELECT deptno, SUM(sal) FROM emp
WHERE job <> 'MANAGER'
GROUP BY deptno
ORDER BY SUM(sal) DESC
```

```
LIMIT 1;

SELECT deptno, SUM(sal) FROM emp
GROUP BY deptno
WHERE job <> 'MANAGER'
ORDER BY SUM(sal) DESC
LIMIT 1; -- error: SELECT (1) WHERE (2) GROUP BY (3) ORDER BY (4) LIMIT
```

Having clause

- To filter output of GROUP BY.
- HAVING clause MUST be written After GROUP BY.
- HAVING clause
  - filtering based on some condition.
  - filter output of GROUP BY.
  - condition on aggregate function.
  - must be after the GROUP BY.
- WHERE clause
  - filtering based on some condition.
  - filter rows of table.
  - condition on individual columns.
  - if present, must be before the GROUP BY.

```
-- find all jobs who have more than 3 empoyees.
SELECT job, COUNT(empno) FROM emp
GROUP BY job;

SELECT job, COUNT(empno) FROM emp
GROUP BY job
HAVING COUNT(empno) > 3;

-- find all depts who have avg sal more than 2500.0
SELECT deptno, AVG(sal) FROM emp
GROUP BY deptno;

SELECT deptno, AVG(sal) FROM emp
GROUP BY deptno
HAVING AVG(sal) > 2500.0;

-- get total sal of managers, analysts and clerks in deptwise for dept 10 & 20.
SELECT deptno, SUM(sal) FROM emp
WHERE job IN ('MANAGER', 'ANALYST', 'CLERK')
GROUP BY deptno;

SELECT deptno, SUM(sal) FROM emp
WHERE job IN ('MANAGER', 'ANALYST', 'CLERK')
GROUP BY deptno
HAVING deptno IN (10, 20);

SELECT deptno, SUM(sal) FROM emp
```

```
WHERE job IN ('MANAGER', 'ANALYST', 'CLERK') AND deptno IN (10, 20)
GROUP BY deptno;

-- find num of emps per job for MANAGER and ANALYST.
SELECT job, COUNT(empno) FROM emp
GROUP BY job
HAVING job IN ('MANAGER', 'ANALYST');

SELECT job, COUNT(empno) FROM emp
WHERE job IN ('MANAGER', 'ANALYST')
GROUP BY job;

-- find jobs where avg sal < 2000.
SELECT job, AVG(sal) FROM emp
WHERE AVG(sal) < 2000
GROUP BY job; -- error

SELECT job, AVG(sal) FROM emp
GROUP BY job
HAVING AVG(sal) < 2000;
```

```
SELECT ename, SUM(sal) FROM emp
GROUP BY ename;

SELECT ename, sal FROM emp;
```

REGEXP operator

- Used in WHERE clause to filter the records.
- Similar to LIKE to filter based on some pattern (of string type).
    - Wildcard chars: %, _

```
-- find all emps whose name start with A
SELECT * FROM emp WHERE ename LIKE 'A%';
```

- REGEXP stands for Regular Expressions. Used at many places.
    - Database -- for searching/validation.
        - RDBMS, NoSQL
    - Programming languages
        - Python, JS, Java, C#, Perl, ...
    - Web Programming -- for validation
        - HTML, JS, PHP, Java, Python, .NET, ...
- REGEXP operator allows giving patterns using rich Wildcard.
    - Wildcard chars: $, ^, ., +, *, [], [^], {}, ...

- ^ <-- at the start.
- $ <-- at the end.
- . <-- any one char.
- [a-z] <-- scanset/range e.g. a to z
- [a-z0-9] <-- scanset/range e.g. a to z or 0 to 9
- [aiu] <-- scanset -- any one letter out of a, i and u
- [^...] <-- inverse of scanset
  - [^a-z] <-- anything other than alphabet
  - [^0-9] <-- anything other than digit
- (...) <-- group multiple chars

```
CREATE TABLE food(word VARCHAR(30));

INSERT INTO food VALUES ('this'), ('biscuit'), ('isnt'), ('tasty'), ('but'),
('that'), ('cake'), ('is'), ('really good');

SELECT * FROM food;

-- get all words containing "is"
SELECT * FROM food WHERE word REGEXP 'is';

-- get all words starting with "is"
SELECT * FROM food WHERE word REGEXP '^is';

-- get all words ending with "is"
SELECT * FROM food WHERE word REGEXP 'is$';

-- get all words having only is.
SELECT * FROM food WHERE word REGEXP '^is$';
```

```
CREATE TABLE selection(word VARCHAR(30));

INSERT INTO selection VALUES ('bag'), ('beg'), ('big'), ('bog'), ('bug'), ('b*g'),
('bg'), ('xyz');

SELECT * FROM selection;

-- get all words having any one char between b and g.
SELECT * FROM selection WHERE word REGEXP 'b.g';

-- get all words having any one alphabet between b and g.
SELECT * FROM selection WHERE word REGEXP 'b[a-z]g';

-- get all words having any one letter between b and g (but not alphabet).
SELECT * FROM selection WHERE word NOT REGEXP 'b[a-z]g';

SELECT * FROM selection WHERE word REGEXP 'b[^a-z]g';

-- get all words having any one letter between b and g out of a, i or u
```

```
SELECT * FROM selection WHERE word REGEXP 'b[aiu]g';

-- get all words 'b*g'
SELECT * FROM selection WHERE word REGEXP 'b*g'; -- wrong out
SELECT * FROM selection WHERE word REGEXP 'b\\*g'; -- \\ removes special meaning
of *
SELECT * FROM selection WHERE word REGEXP 'b[*]g';
```

```
CREATE TABLE repetition(word VARCHAR(40));

INSERT INTO repetition VALUES ('ww'), ('wow'), ('woow'), ('wooow'), ('woooow'),
('wooooow'), ('woooooow'), ('wooooooow');

SELECT * FROM repetition;

-- * means 0 or more occurrences of prev char.
SELECT * FROM repetition WHERE word REGEXP 'wo*w';

-- ? means 0 or 1 occurrence of prev char.
SELECT * FROM repetition WHERE word REGEXP 'wo?w';

-- + means 1 or more occurrences of prev char.
SELECT * FROM repetition WHERE word REGEXP 'wo+w';

-- {n} means n occurrences of prev char.
SELECT * FROM repetition WHERE word REGEXP 'wo{4}w';

-- {m,} means more than m occurrences of prev char.
SELECT * FROM repetition WHERE word REGEXP 'wo{4,}w';

-- {m,n} means m to n occurrences of prev char.
SELECT * FROM repetition WHERE word REGEXP 'wo{3,6}w';
```

```
CREATE TABLE people (id INT, name VARCHAR(20), email VARCHAR(40), phone
VARCHAR(14));

-- get all records where phone numbers are valid
--      10 digits -- 9876543210 --> ^[0-9]{10}$
--      0 can be before 10 digit -- 09876543210 --> ^0?[0-9]{10}$
--      +91 can be before 10 digit -- +919876543210 --> ^(\\+91)?[0-9]{10}$ --> ?
check if (+91) group is present/absent
--      0 or +91 can be before 10 digit --> 09876543210 or +919876543210
--                      --> '^(0|\\+91)?[0-9]{10}$'
```

MySQL RDBMS

Trainer: Mr. Nilesh Ghule

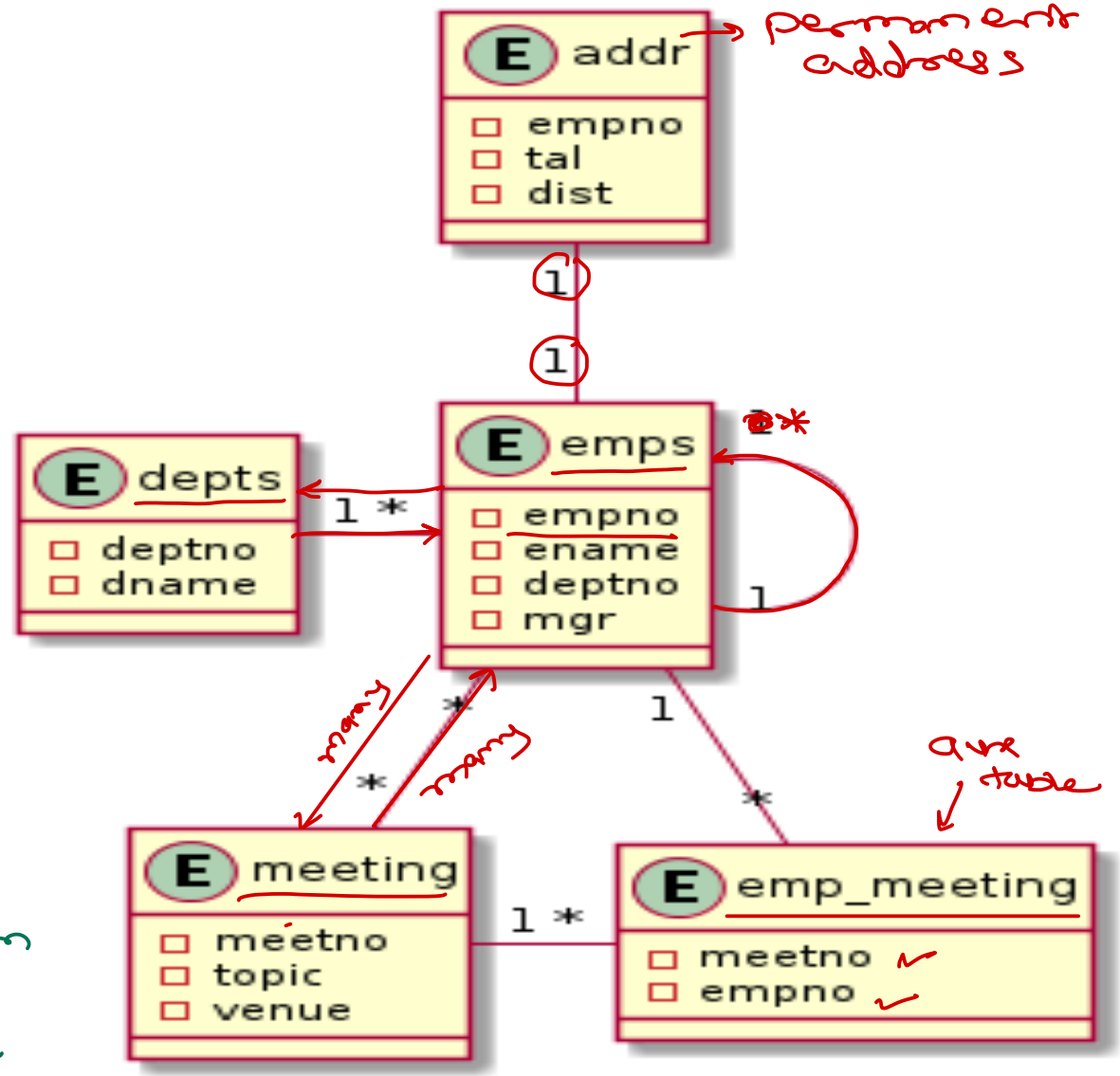
Entity Relations

ER - diagram

- 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
  - One to Many
  - Many to One
  - Many to Many
- Entity relations is outcome of Normalization process.

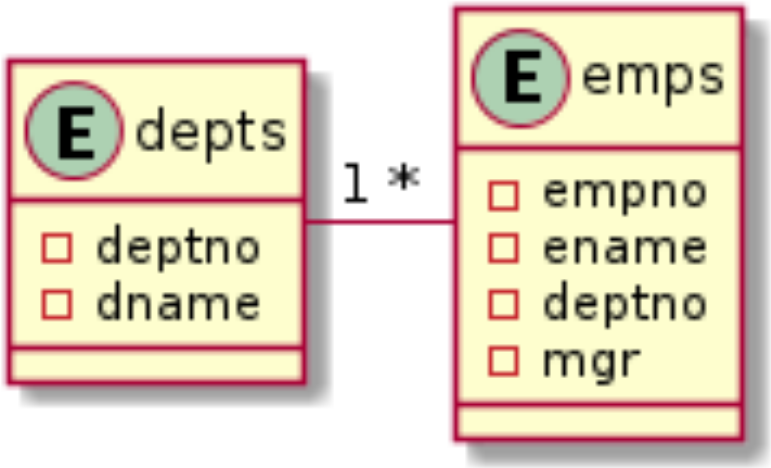
part of application design

1. req analysis
2. design → ui design, dto design, oo design
3. implementation
4. testing / maintence



SQL Joins

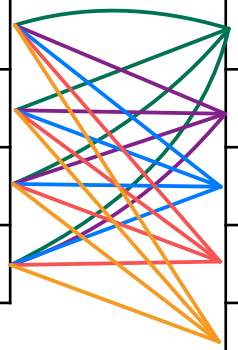
- Join statements are used to SELECT data from multiple tables using single query.
- Typical RDBMS supports following types of joins:
  - Cross Join
  - Inner Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join
  - Self join



Cross Join

depts (4)

deptno	dname
10	DEV
20	QA
30	OPS
40	ACC



(5) emps

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

outer (depts)

```
for (emp e : emps) {
    for (dept d : depts) {
        print (e.ename, d.dname);
    }
}
```

inner (depts)

```
select e.ename, d.dname from emps e
cross join depts d;
```

- 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 → common rows

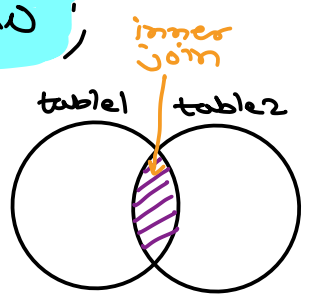
deptno	dname
10	DEV
20	QA
30	OPS
40	ACC

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

```
for(emp e : emps) {
    for(dept d : depts) {
        if(e.deptno == d.deptno)
            print(e.ename, d.dname);
    }
}
```

Amit DEV
Rahul DEV
Nilesh QA

```
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

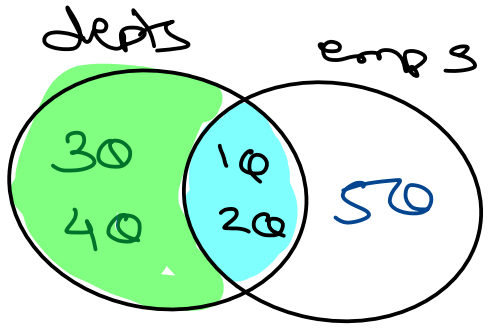
Trainer: Mr. Nilesh Ghule

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

```
for (dept d : depts) {
    found = 0;
    for (emp e : emps) {
        if (d.deptno == e.deptno) {
            print(e.ename, d.dname);
            found = 1;
        }
    }
    if (found == 0)
        print(NULL, d.dname);
}
```




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

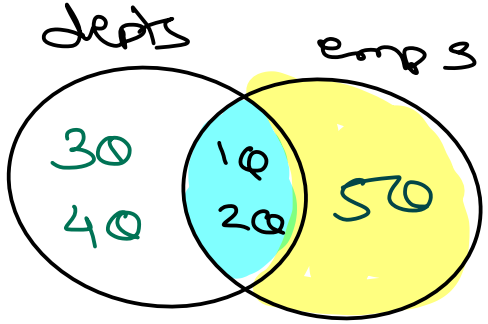
- Left outer join is used to return matching rows from both tables along with additional rows in left table.
- Corresponding to additional rows in left table, right table values are taken as NULL.
- OUTER keyword is optional.

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

```
for(emp e : emps) {
  found = 0;
  for (dept d : depts){
    if(d.deptno == e.deptno) {
      print(e.ename, d.dname);
      found = 1;
    }
  }
  if (found == 0)
    print(e.ename, NULL);
}
```



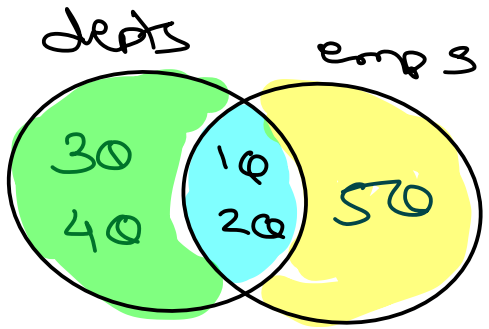

```
select e.ename, d.dname from depts d
right outer join emps e on d.deptno = e.deptno;
```

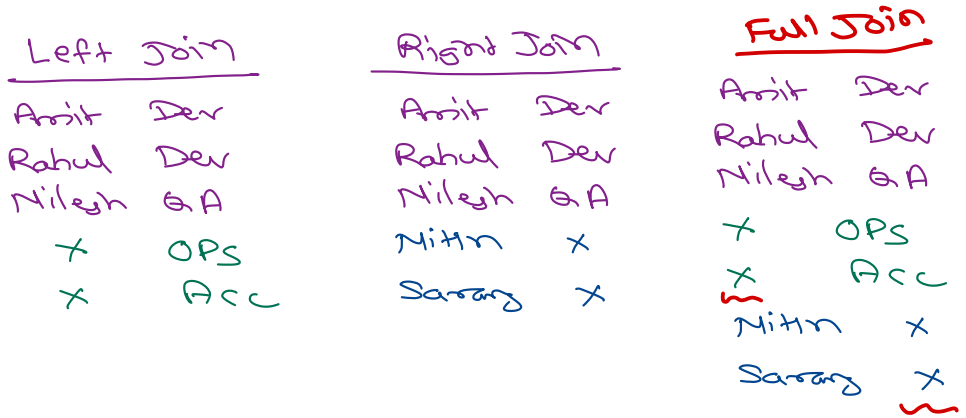
- Right outer join is used to return matching rows from both tables along with additional rows in right table.
- Corresponding to additional rows in right table, left table values are taken as NULL.
- OUTER keyword is optional.

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 – Combine output of two queries.

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

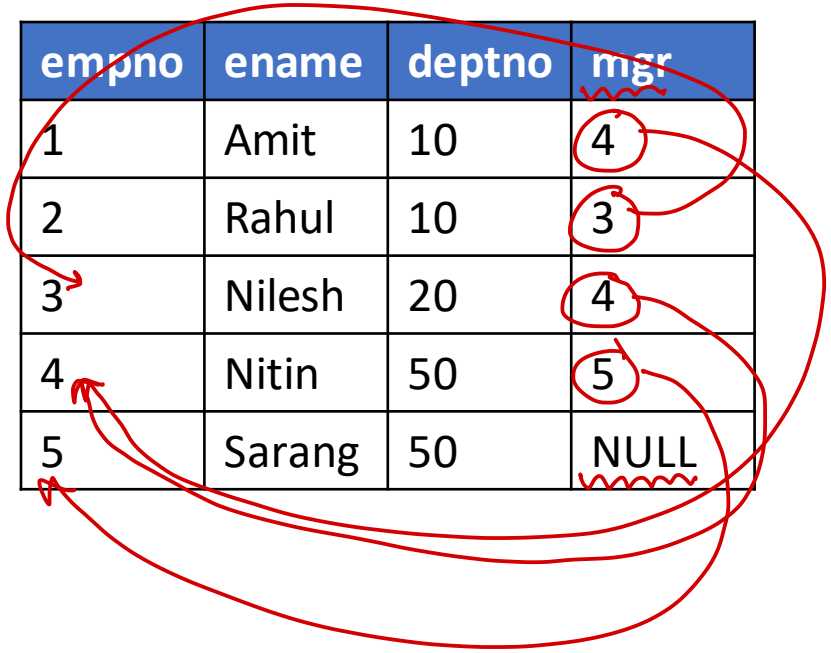
ename	dname
Amit	DEV
Rahul	DEV
Nilesh	QA
Nitin	NULL
Sarang	NULL

- 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



mgr column - represent empno of manager of current emp.

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

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

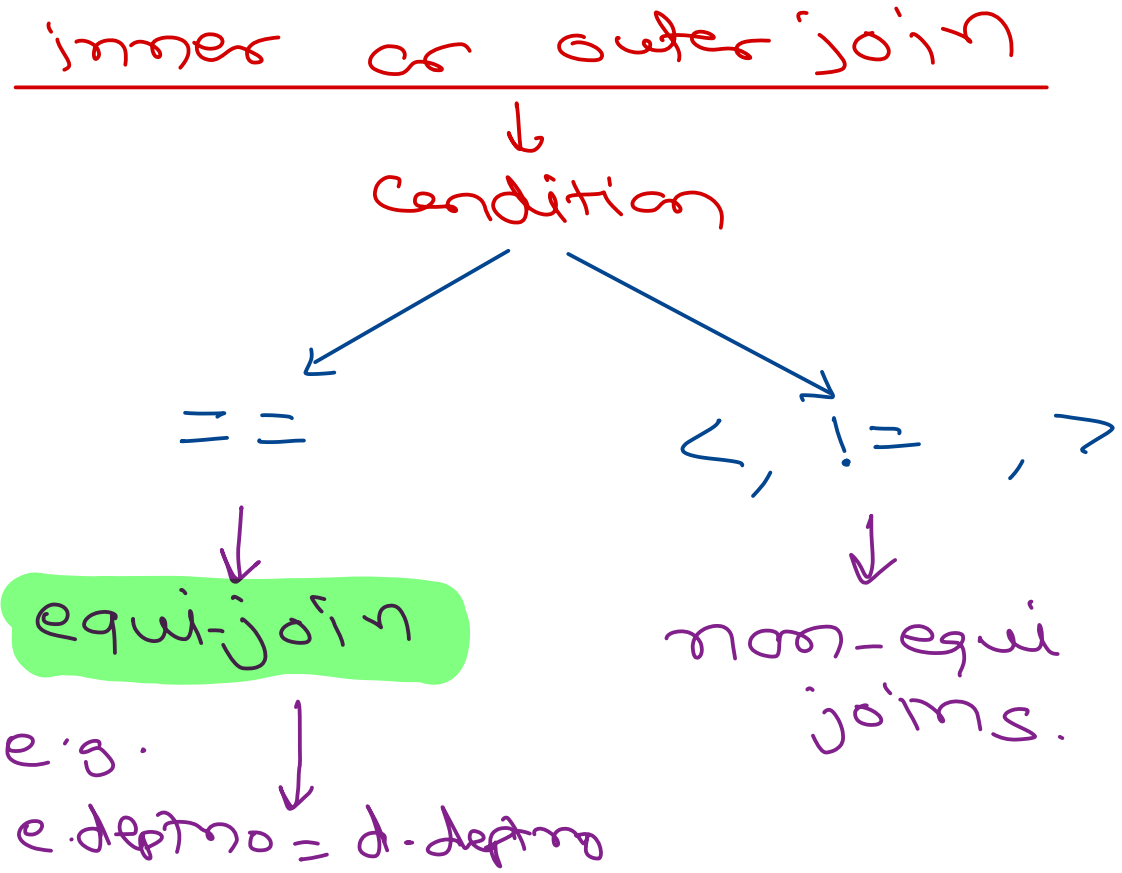
empno	ename	deptno	mgr
1	Amit	10	4
2	Rahul	10	3
3	Nilesh	20	4
4	Nitin	50	5
5	Sarang	50	NULL

e | m
Amit, Nitin
Rahul, Nilesh
Nilesh, Nitin
Nitin, Sarang

e | m
Amit, Nitin
Rahul, Nilesh
Nilesh, Nitin
Nitin, Sarang
Sarang, x

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

Joins




```
select e.ename, d.dname from
depts d inner join emps e
on e.deptno = d.deptno;
```

```
select e.ename, d.dname from
depts d join emps e
using (deptno);
```

→ col name is same in both tables.

Joins – Inner join

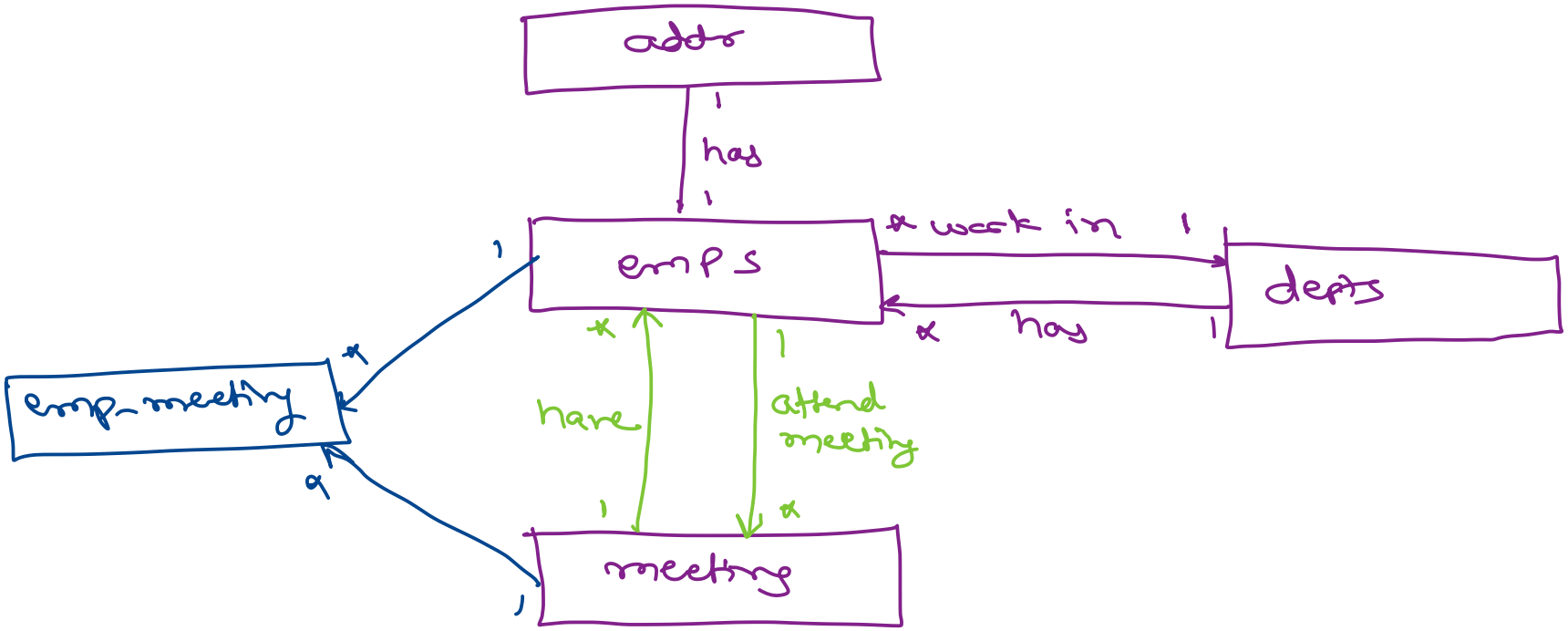
standard

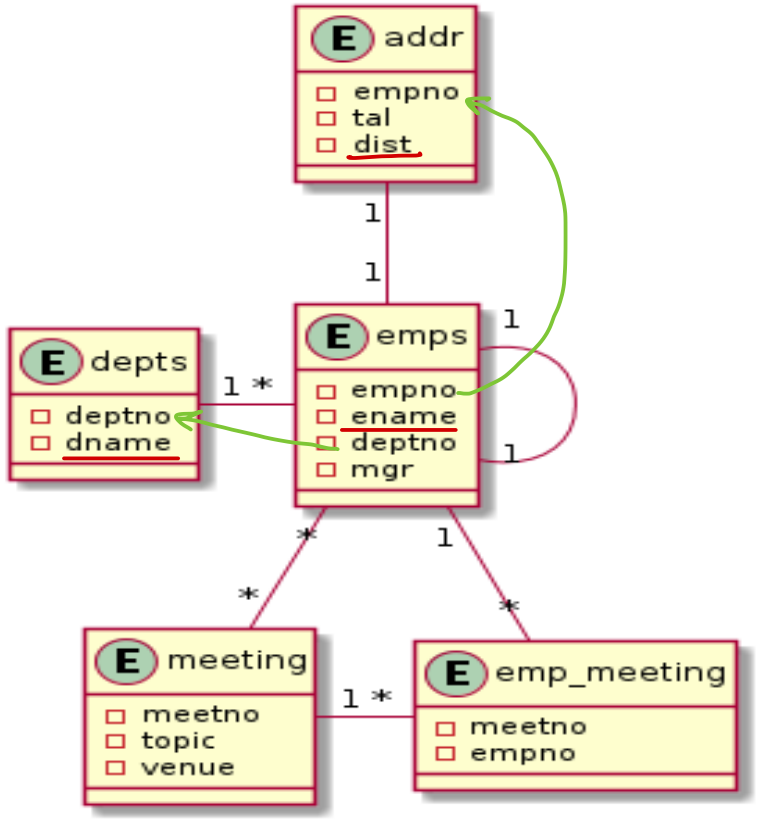
```
select e.ename, d.dname from
emps e inner join depts d
on e.deptno = d.deptno;
```

non-standard

```
select e.ename, d.dname from
emps e, depts d
where e.deptno = d.deptno;
```

Joins


addr
has
emps
work in
depts
has
emp_meeting
have
attend meeting
meeting


addr
empno
tal
dist
emps
empno
ename
deptno
mgr
depts
deptno
dname
meeting
meetno
topic
venue
emp_meeting
meetno
empno

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Agenda

- Table Relations
- Joins
- ~~Views~~
- ~~Constraints~~
- ~~ALTER~~

Table Relations

- One To One
- One To Many
- Many To One
- Many To Many

```
Emp Table
empno       name            deptno
1           Nitin           10
2           Nilesh          20
3           Amit            20
4           Yogesh          20

Meeting Table
meetno      Topic                   Venue
101         May21 Batch Planning    Zoom
102         Audit                   Skype

Emp_Meeting Table (Aux)
empno       meetno
1           101
2           101
4           101
1           102
2           102
3           102
```


Joins

Cross Join

- SELECT columns FROM table1 t1 CROSS JOIN table2 t2;

```
SELECT e.ename, d.dname FROM emps e
CROSS JOIN depts d;
```

```
SELECT e.ename, d.dname FROM depts d
CROSS JOIN emps e;

SELECT emps.ename, depts.dname FROM depts
CROSS JOIN emps;

SELECT ename, dname FROM depts
CROSS JOIN emps;
```

Inner Join

- SELECT columns FROM table1 t1 INNER JOIN table2 t2 ON condition;

```
SELECT e.ename, d.dname FROM emps e
CROSS JOIN depts d
WHERE e.deptno = d.deptno;
-- will work, but not good practice -- instead use inner join -- it is designed
for matching rows

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

SELECT e.ename, d.dname FROM emps e
INNER JOIN depts d
WHERE e.deptno = d.deptno;
-- will work, but not good practice -- ON clause is defined to given condition on
join.

SELECT e.ename, d.dname FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno
WHERE e.ename IN ('Amit', 'Nilesh');
-- where should be used to filter output of join.
```

Outer Join

- Common rows + Extra rows in the table(s)

Left Outer Join

- Common rows + Extra rows in the left table
- Left table <-- Table appeared before JOIN keyword.

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

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

Right Outer Join

- Common rows + Extra rows in the right table
- Right table <-- Table appeared after JOIN keyword.

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

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

Full Outer Join

- Common rows + Extra rows in the both table

```
SELECT e.ename, d.dname FROM depts d
FULL OUTER JOIN emps e ON e.deptno = d.deptno;
-- error in MySQL -- not supported.
```

Self Join

```
SELECT e.ename, m.ename FROM emps e
INNER JOIN emps m ON e.mgr = m.empno;

SELECT e.ename, m.ename FROM emps e
LEFT JOIN emps m ON e.mgr = m.empno;
```

USING keyword

- If column names on which join condition is given is same in both tables, then we can use USING keyword to short-hand the join condition.
- USING keyword can only be used for equi-join i.e. equality condition in join.

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

SELECT e.ename, d.dname FROM depts d
INNER JOIN emps e USING(deptno);

SELECT e.ename, d.dname FROM depts d
LEFT JOIN emps e USING(deptno);

SELECT e.ename, d.dname FROM depts d
```

```
RIGHT JOIN emps e USING(deptno);
```

Union Operators

- (query1) UNION ALL (query2);
  - Common results are duplicated.
- (query1) UNION (query2);
  - Common results are not duplicated.

```
SELECT ename FROM emp WHERE job = 'CLERK';

SELECT ename FROM emp WHERE deptno = 30;

(SELECT ename FROM emp WHERE job = 'CLERK')
UNION ALL
(SELECT ename FROM emp WHERE deptno = 30);

(SELECT ename FROM emp WHERE job = 'CLERK')
UNION
(SELECT ename FROM emp WHERE deptno = 30);

(SELECT ename FROM emp)
UNION
(SELECT dname FROM dept);

(SELECT ename, sal, job FROM emp)
UNION
(SELECT dname FROM dept);
-- error: two select statements have different number of columns

(SELECT sal FROM emp)
UNION
(SELECT dname FROM dept);
```

```
(SELECT e.ename, d.dname FROM depts d
LEFT OUTER JOIN emps e ON e.deptno = d.deptno)
UNION ALL
(SELECT e.ename, d.dname FROM depts d
RIGHT OUTER JOIN emps e ON e.deptno = d.deptno);

(SELECT e.ename, d.dname FROM depts d
LEFT OUTER JOIN emps e ON e.deptno = d.deptno)
UNION
(SELECT e.ename, d.dname FROM depts d
RIGHT OUTER JOIN emps e ON e.deptno = d.deptno);
```

Advanced Joins

```
SELECT e.ename, d.dname FROM emps e, depts d
WHERE e.deptno = d.deptno;
```

```
-- print deptname and dist of all emps.
SELECT e.ename, d.dname FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno;

SELECT e.ename, d.dname, a.dist FROM emps e
INNER JOIN depts d ON e.deptno = d.deptno
INNER JOIN addr a ON e.empno = a.empno;

SELECT e.ename, d.dname, a.dist FROM emps e
LEFT JOIN depts d ON e.deptno = d.deptno
INNER JOIN addr a ON e.empno = a.empno;
```

```
-- print emp name and meeting topic they are attending
SELECT e.ename, em.meetno FROM emp_meeting em
INNER JOIN emps e ON em.empno = e.empno;

SELECT e.ename, em.meetno, m.topic FROM emp_meeting em
INNER JOIN emps e ON em.empno = e.empno
INNER JOIN meeting m ON em.meetno = m.meetno;

-- emps are travelling for meeting from their hometown.
-- "also" print dist from where they are coming to meeting.
SELECT e.ename, em.meetno, m.topic, a.dist FROM emp_meeting em
INNER JOIN emps e ON em.empno = e.empno
INNER JOIN meeting m ON em.meetno = m.meetno
INNER JOIN addr a ON e.empno = a.empno;

-- emps are also representing their depts.
-- "also" print dnames in which they are working.
SELECT e.ename, em.meetno, m.topic, a.dist, d.dname FROM emp_meeting em
INNER JOIN emps e ON em.empno = e.empno
INNER JOIN meeting m ON em.meetno = m.meetno
INNER JOIN addr a ON e.empno = a.empno
LEFT JOIN depts d ON e.deptno = d.deptno;
```

```
-- (old classwork) print dname and num of emps working in each dept.
SELECT deptno, COUNT(empno) FROM emp
GROUP BY deptno;

SELECT d.dname, COUNT(e.empno) FROM emp e
INNER JOIN dept d ON e.deptno = d.deptno;
```

```
-- error: group fns need group by

SELECT d.dname, COUNT(e.empno) FROM emp e
INNER JOIN dept d ON e.deptno = d.deptno
GROUP BY e.deptno;
-- error: columns to be selected must be in group by

SELECT d.dname, COUNT(e.empno) FROM emp e
INNER JOIN dept d ON e.deptno = d.deptno
GROUP BY d.dname;

SELECT d.dname, COUNT(e.empno) FROM emp e
RIGHT JOIN dept d ON e.deptno = d.deptno
GROUP BY d.dname;

-- print dname and empcount in desc order of emp count.
SELECT d.dname, COUNT(e.empno) FROM emp e
RIGHT JOIN dept d ON e.deptno = d.deptno
GROUP BY d.dname
ORDER BY COUNT(e.empno) DESC;

-- print dname and empcount of dept having highest emps.
SELECT d.dname, COUNT(e.empno) FROM emp e
RIGHT JOIN dept d ON e.deptno = d.deptno
GROUP BY d.dname
ORDER BY COUNT(e.empno) DESC
LIMIT 1;

-- print dname and clerk count of dept having highest clerk.
SELECT d.dname, COUNT(e.empno) FROM emp e
RIGHT JOIN dept d ON e.deptno = d.deptno
WHERE job = 'CLERK'
GROUP BY d.dname
ORDER BY COUNT(e.empno) DESC
LIMIT 1;
```

Joins Syntax

```
SELECT columns FROM table1
XXX JOIN table2 ON condition1
XXX JOIN table3 ON condition2
WHERE condition
GROUP BY column
HAVING condition
ORDER BY column ASC/DESC
LIMIT m,n;
```

SQL

- SQL is Declarative Query Language

- Programmer decides what to do, but doesn't define how to do.
- Internal implementation depends on RDBMS (may differ in different RDBMS).

MySQL - RDBMS

Trainer: Mr. Nilesh Ghule

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.

where ✓
having ✓

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 of value.
    - ANY operator compares with one of the result from sub-queries. → like OR
    - ALL operator compares with all the results from sub-queries. → like AND

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Trainer: Mr. Nilesh Ghule

Sub queries

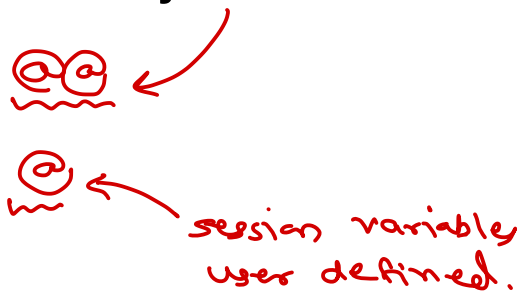
- Correlated sub-query
  - If number of results from sub-query are reduced, query performance will increase.
  - This can be done by adding criteria (WHERE clause) in sub-query based on outer query row.
  - Typically correlated sub-query use IN, ALL, ANY and EXISTS operators.

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.

Query performance → RDBMS specific.

- 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

Agenda

- Sub-queries
- Views
- ALTER
- Indexes
- Constraints

Q & A

```
SELECT emp.ename, dept.dname FROM emp
INNER JOIN dept ON emp.deptno = dept.deptno;

SELECT ename, dname FROM emp
INNER JOIN dept ON emp.deptno = dept.deptno;
```

```
SELECT e.ename, d.dname FROM emp e
INNER JOIN dept d ON e.deptno = d.deptno;

SELECT e.ename, d.dname FROM emp e
INNER JOIN dept d USING (deptno);
-- join on common column -- deptno.

SELECT e.ename, d.dname FROM emp e
NATURAL JOIN dept d;
-- inner join on common column -- names of common columns are found automatically.
```

Sub-queries

Single-Row Sub-query

```
-- find emp with max sal.
SELECT * FROM emp WHERE sal = MAX(sal);
-- error: group fn cannot be used in WHERE clause

SET @maxsal = (SELECT MAX(sal) FROM emp);
-- get output of max sal query into a variable @maxsal.
SELECT @maxsal;
-- print the variable @maxsal
SELECT * FROM emp WHERE sal = @maxsal;
-- use the variable in where clause to get desired result.

SELECT * FROM emp WHERE sal = (SELECT MAX(sal) FROM emp);
```

```
-- instead of a var using a query in another query -- sub-query

-- find the last hired employee in company.
SELECT * FROM emp
ORDER BY hire DESC
LIMIT 1;

SET @lasthire = (SELECT MAX(hire) FROM emp);
SELECT @lasthire;
SELECT * FROM emp WHERE hire = @lasthire;

SELECT * FROM emp WHERE hire = (SELECT MAX(hire) FROM emp);
```

```
-- find emp with highest sal.
SELECT * FROM emp ORDER BY sal DESC LIMIT 1;

SELECT * FROM emp WHERE sal = (SELECT MAX(sal) FROM emp);

-- find emp with 3rd highest sal.
SELECT * FROM emp ORDER BY sal DESC;

SELECT * FROM emp ORDER BY sal DESC LIMIT 2, 1;

SELECT DISTINCT sal FROM emp ORDER BY sal DESC;

SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT 2,1;

SET @sal3 = (SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT 2,1);
SELECT @sal3;
SELECT * FROM emp WHERE sal = @sal3;

SELECT * FROM emp WHERE sal = (SELECT DISTINCT sal FROM emp ORDER BY sal DESC
LIMIT 2,1);

-- find emp with 2nd highest sal.
SET @sal2 = (SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT 1,1);
SELECT @sal2;
SELECT * FROM emp WHERE sal = @sal2;

SELECT * FROM emp WHERE sal = (SELECT DISTINCT sal FROM emp ORDER BY sal DESC
LIMIT 1,1);

-- find all emps working in dept of JAMES.
SELECT deptno FROM emp WHERE ename = 'JAMES';

SET @james_dept=(SELECT deptno FROM emp WHERE ename = 'JAMES');
SELECT * FROM emp WHERE deptno = @james_dept;

SELECT * FROM emp
WHERE deptno = (SELECT deptno FROM emp WHERE ename = 'JAMES');
-- WHERE clause & hence sub-query is executed once for each record.
```

```
-- For emp table it will be executed 14 times (because there are 14 rows).

-- However most of RDBMS have optimizations, which ensure that sub-query result is
cached (when appropriate)
-- This execute sub-query only once and hence much efficient,
-- This depends on RDBMS optimizations.

SELECT e.ename, e.deptno FROM emp e
INNER JOIN emp j ON e.deptno = j.deptno
WHERE j.ename = 'JAMES';

-- optimization settings of MySQL can be seen in optimizer_switch variable.
SELECT @@optimizer_switch;
```

Single-Row Sub-query

```
-- find all emps having sal more than sals of salesman.

-- using single row sub-query
SELECT MAX(sal) FROM emp WHERE job = 'SALESMAN';

SELECT * FROM emp
WHERE sal > (SELECT MAX(sal) FROM emp WHERE job = 'SALESMAN');

-- using multi-row sub-query
SELECT sal FROM emp WHERE job = 'SALESMAN';

SELECT * FROM emp
WHERE sal > (SELECT sal FROM emp WHERE job = 'SALESMAN');
-- error: left side of > have one value, right side of > have 4 values.
-- multi-row sub-query results cannot be compared using relational operators

SELECT * FROM emp
WHERE sal > ALL(SELECT sal FROM emp WHERE job = 'SALESMAN');
-- ALL keyword -- compare with all values.
```

```
-- find all emps having sal less than than any of the salesman.

SELECT sal FROM emp WHERE job = 'SALESMAN';

-- using single row sub-query
SELECT MAX(sal) FROM emp WHERE job = 'SALESMAN';

SELECT * FROM emp
WHERE sal < (SELECT MAX(sal) FROM emp WHERE job = 'SALESMAN');

-- using multi-row sub-query
SELECT sal FROM emp WHERE job = 'SALESMAN';
```

```
SELECT * FROM emp
WHERE sal < ANY(SELECT sal FROM emp WHERE job = 'SALESMAN');
-- ANY keyword -- compare with all values and find if sal is less than any of
them.

-- find all emps whose sal is less than sal of all salesman.
SELECT * FROM emp
WHERE sal < ALL(SELECT sal FROM emp WHERE job = 'SALESMAN');
```

```
-- find all depts which have some emp.

SELECT * FROM emp;

SELECT * FROM dept;

SELECT deptno FROM emp;

SELECT * FROM dept
WHERE deptno = (SELECT deptno FROM emp);
-- error: left side have one deptno and right side have 14 deptno

SELECT * FROM dept
WHERE deptno = ANY(SELECT deptno FROM emp);

SELECT * FROM dept
WHERE deptno IN (SELECT deptno FROM emp);
```

Multi-Row Sub-query Operators

- IN operator / NOT IN operator
  - To check equality in multi-row sub-queries.
- ALL operator
  - comparing with all values and if condition is true for all of them, then consider it true.
  - similar to logical AND.
  - SELECT sal FROM emp WHERE job = 'SALESMAN';
    - 1600, 1250, 1250, 1500.
  - SELECT * FROM emp WHERE sal > ALL(SELECT sal FROM emp WHERE job = 'SALESMAN');
    - sal > 1600 AND sal > 1250 AND sal > 1250 AND sal > 1500.
- ANY operator
  - comparing with all values and if condition is true for any of them, then consider it true.
  - similar to logical OR.
  - SELECT sal FROM emp WHERE job = 'SALESMAN';
    - 1600, 1250, 1250, 1500.
  - SELECT * FROM emp WHERE sal > ANY(SELECT sal FROM emp WHERE job = 'SALESMAN');
    - sal > 1600 OR sal > 1250 OR sal > 1250 OR sal > 1500.

Correlated Sub-query

- Inner query is executed for each row of outer query.
- To improve performace of sub-queries, the inner query should return minimum number of rows.

```
SELECT * FROM dept
WHERE deptno IN (SELECT deptno FROM emp);
-- since outer query has 4 rows, inner query will be executed 4 times.

SELECT * FROM dept
WHERE deptno IN (SELECT DISTINCT deptno FROM emp);
-- even though sub-query returns only 3 rows, but it need to process all 14 rows
to find unique.

SELECT deptno FROM emp WHERE deptno = 10; --> 3 rows -- 10, 10, 10
SELECT deptno FROM emp WHERE deptno = 20; --> 5 rows -- 20, 20, 20, 20, 20
SELECT deptno FROM emp WHERE deptno = 30; --> 6 rows -- 30, 30, 30, 30, 30, 30

SELECT * FROM dept d
WHERE d.deptno IN (SELECT e.deptno FROM emp e);
-- still inner query returns 14 rows only.
-- dept table have 4 rows, so sub-query executed 4 times and returns 14 rows each
times.

SELECT * FROM dept d
WHERE d.deptno IN (SELECT e.deptno FROM emp e WHERE e.deptno = d.deptno);
-- dept table have 4 rows, so sub-query executed 4 times.
--      SELECT e.deptno FROM emp e WHERE e.deptno = 10; -- 3 rows
--      SELECT e.deptno FROM emp e WHERE e.deptno = 20; -- 5 rows
--      SELECT e.deptno FROM emp e WHERE e.deptno = 30; -- 6 rows
--      SELECT e.deptno FROM emp e WHERE e.deptno = 40; -- 0 rows
-- here sub-query returns less rows, hence better performance.
-- condition is based on current row of outer query.

SELECT * FROM dept d
WHERE EXISTS (SELECT e.deptno FROM emp e WHERE e.deptno = d.deptno);
-- exists only check if inner query result is non-empty.

-- find all depts in which there are no employees
SELECT * FROM dept d
WHERE d.deptno NOT IN (SELECT e.deptno FROM emp e WHERE e.deptno = d.deptno);

SELECT * FROM dept d
WHERE NOT EXISTS (SELECT e.deptno FROM emp e WHERE e.deptno = d.deptno);
-- exists only check if inner query result is empty.
```

Sub-query with DML

```
-- delete emp with max sal.
DELETE FROM emp WHERE sal = MAX(sal);
-- error: group function in WHERE clause is not allowed.
```

```
DELETE FROM emp
WHERE sal = (SELECT MAX(sal) FROM emp);
-- error: not supported in MySQL.
-- MySQL doesn't allow inner query to be performed on the table, in which
UPDATE/DELETE operation is going on.

SET @maxsal = (SELECT MAX(sal) FROM emp);
DELETE FROM emp WHERE sal = @maxsal;

SELECT * FROM emp;

-- delete dept in which there is no emp.
DELETE FROM dept
WHERE deptno NOT IN (SELECT deptno FROM emp);
-- alowed in MySQL
-- Table to delete from: "dept", Table on which inner query is running: "emp".

SELECT * FROM dept;

-- insert JOHN in RESEARCH dept as ANALYST on sal 2800.
INSERT INTO emp(empno, ename, job, sal, deptno)
VALUES (1000, 'JOHN', 'ANALYST', 2800, 20);

INSERT INTO emp(empno, ename, job, sal, deptno)
VALUES (1000, 'JOHN', 'ANALYST', 2800,
    (SELECT deptno FROM dept WHERE dname='RESEARCH')
);

SELECT * FROM emp;

-- insert JOHN in RESEARCH dept as ANALYST on with sal same as highest ANALYST
sal.
-- keep its empno as MAX(empno) + 1.

INSERT INTO emp(empno, ename, job, sal, deptno)
VALUES (
    (SELECT MAX(empno) FROM emp)+1,
    'JOHN',
    'ANALYST',
    (SELECT MAX(sal) FROM emp WHERE job = 'ANALYST'),
    (SELECT deptno FROM dept WHERE dname='RESEARCH')
);
-- error: Not supported in MySQL
-- Homework: Do this using variables
```

Query Performance

```
EXPLAIN FORMAT=JSON
SELECT * FROM dept d
WHERE d.deptno IN (SELECT e.deptno FROM emp e);
```

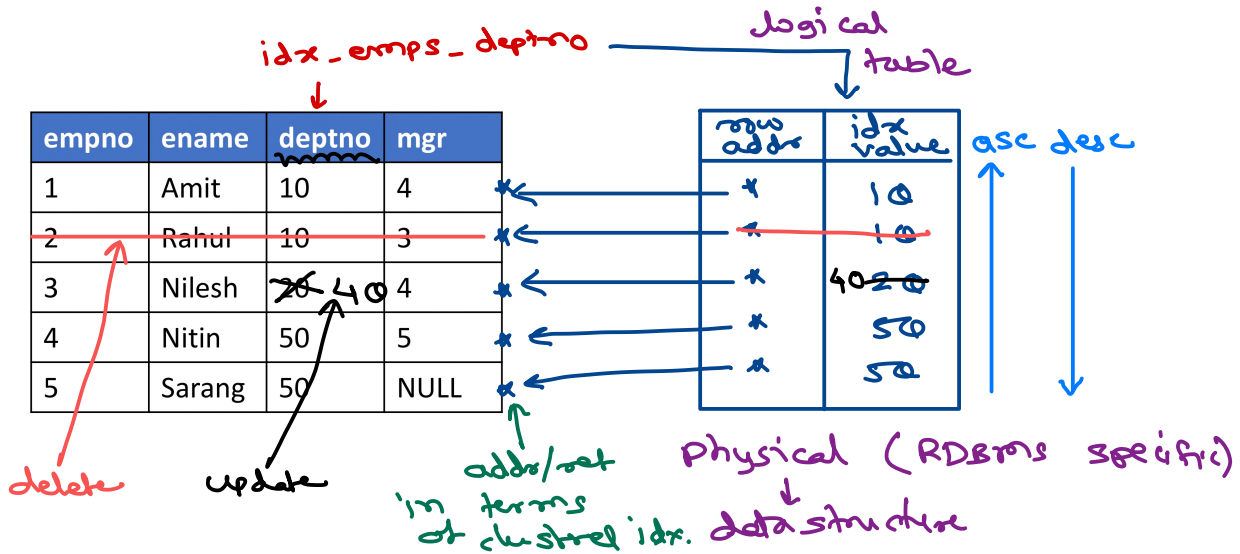
```
-- 4.90

EXPLAIN ANALYZE
SELECT * FROM dept d
WHERE d.deptno IN (SELECT e.deptno FROM emp e);

EXPLAIN FORMAT=JSON
SELECT * FROM dept d
WHERE EXISTS (SELECT e.deptno FROM emp e WHERE e.deptno = d.deptno);
-- 4.90

EXPLAIN ANALYZE
SELECT * FROM dept d
WHERE EXISTS (SELECT e.deptno FROM emp e WHERE e.deptno = d.deptno);
```

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

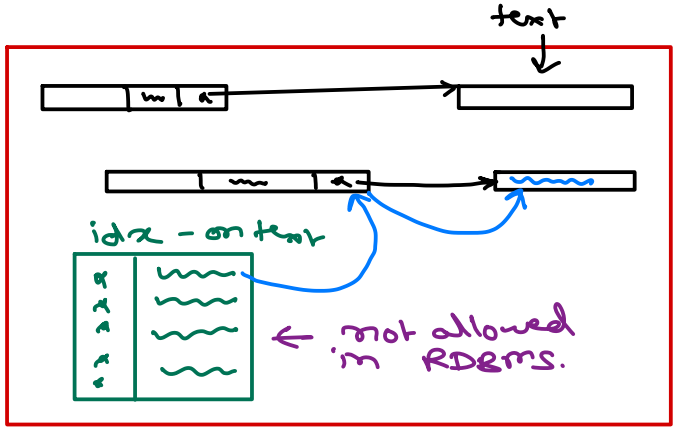
- Index can be ASC or DESC.
  - It cause storage of key values in respective order (MySQL 8.x onwards).
  - ASC/DESC index is used by optimizer on ORDER BY queries.
- There are four types of indexes:
  - Simple index
    - CREATE INDEX idx_name ON table(column [ASC|DESC]);
  - Unique index
    - CREATE 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.

hidden column added by RDBMS

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;

index on first 10 bytes.
i.e. only first 10 bytes used for searching with indexes.

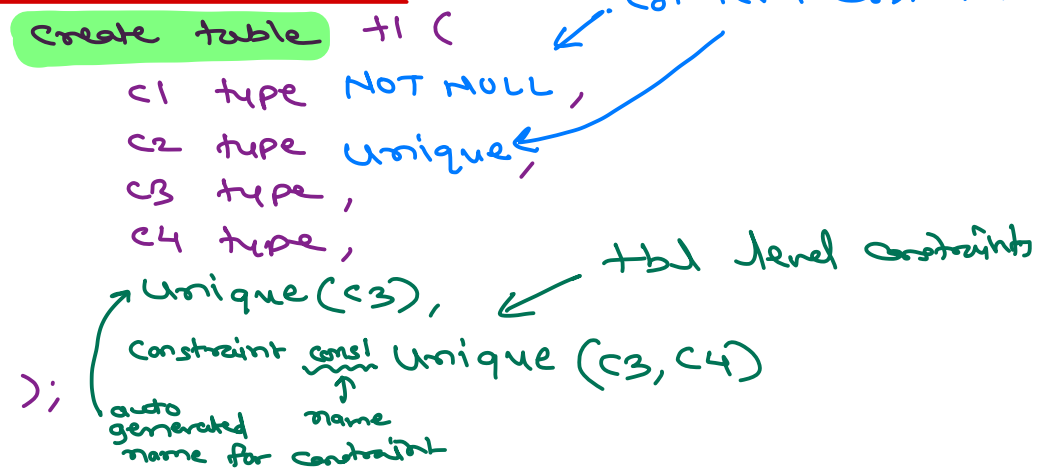


- When table is dropped, all indexes are automatically dropped.
- Indexes should not be created on the columns not used frequent search, ordering or grouping operations.
- Columns in join operation should be indexed for better performance.

emp(deptno) → ON e.deptno = d.deptno dept(deptno)

Constraints

- Constraints are restrictions imposed on columns.
- There are five constraints
  - NOT NULL → col level
  - UNIQUE → col or tbl level
  - PRIMARY KEY → col or tbl level
  - FOREIGN KEY → col or tbl level
  - CHECK → col or tbl level



- 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.

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));

```
Create table students (
    std INT not null,
    roll INT not null,
    name CHAR(20),
    unique (std, roll)
);
```

internally create unique composite index.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Trainer: Mr. Nilesh Ghule

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));

```
create table students (
    std INT not null,
    roll INT not null,
    name CHAR(20),
    unique (std, roll)
);
```

internally create unique composite index.

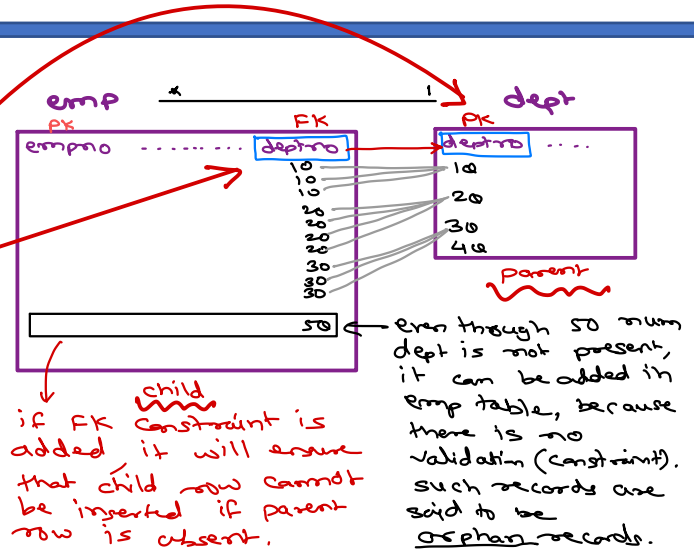
one table may have multiple unique keys.

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.
  - 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".
  - FK can have duplicate values as well as null values.
  - FK constraint is applied on column of child table (not on parent table).
  - parent rows cannot be deleted, until child 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))



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);

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Agenda

- Lab Exam Pattern
- Indexes
- Constraints
- ~~ALTER~~
- ~~Views~~
- ~~DCL~~

Internal assesments

- Rapid fire sheet -- approx "15" questions
  - Study from class notes/lecture.
  - Hand written (organized) answers on your notebook
  - Ensure that your name is written on each page (in your handwriting)
  - Scan (Adobe scanner) and upload as per instructions
  - This sheet is helpful during campus for quick revision.
- Interview question video.
  - 5 mins single video.
  - max 3 questions.
  - Improving presentation skills.
- Assignments/Queries
  - 20 queries
  - Understading problem statement is part of assesments
  - Write the best answer

Lab Exam Pattern

- 90 to 120 mins -- proctored exam
- There will be different database tables.
- SQL queries - SELECT, INSERT, UPDATE, DELETE, GROUP BY, ORDER BY, JOINS, SUB-QUERIES, etc.
- PSM programs - Stored Procedure, Function, Trigger.
- MySQL HELP command is allowed.
- Mock lab exam -- to understand exam submission process.

Indexes

Simple Index

```
SELECT * FROM emps;

SELECT * FROM emps WHERE deptno = 10;

EXPLAIN FORMAT=JSON
```

```
SELECT * FROM emps WHERE deptno = 10;
-- 0.75

CREATE INDEX idx_emps_deptno ON emps(deptno);
-- by default indexes are ascending

SELECT * FROM emps WHERE deptno = 10;

EXPLAIN FORMAT=JSON
SELECT * FROM emps WHERE deptno = 10;
-- 0.70

CREATE INDEX idx_emps_ename ON emps(ename DESC);

SELECT * FROM emps WHERE ename = 'Nitin';

SHOW INDEXES FROM emps;

DESCRIBE emps;
```

Unique Index

```
CREATE UNIQUE INDEX idx_emps_empno ON emps(empno);

SHOW INDEXES FROM emps;
-- non-unique = 0

DESCRIBE emps;

SELECT * FROM emps;

INSERT INTO emps VALUES (3, 'John', 30, 1);
-- error: empno 3 already exists

INSERT INTO emps VALUES (7, 'Motu', 30, 1);
```

Composite Index

- Index on combination of two or more columns.
- WHERE clause on multiple columns execute faster.

```
SELECT * FROM emp;

-- find all CLERK working in dept 20.
SELECT * FROM emp WHERE deptno = 20 AND job = 'CLERK';

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE deptno = 20 AND job = 'CLERK';
-- 1.65
```

```
CREATE INDEX idx_emo_deptno_job ON emp(deptno ASC, job ASC);

SHOW INDEXES FROM emp;
DESCRIBE emp;

SELECT * FROM emp WHERE deptno = 20 AND job = 'CLERK';

EXPLAIN FORMAT=JSON
SELECT * FROM emp WHERE deptno = 20 AND job = 'CLERK';
-- 0.70
```

```
-- roll no are unique in each std.
CREATE TABLE students(roll INT, std INT, name CHAR(30), marks DECIMAL(5,2));

INSERT INTO students VALUES (1, 1, 'Soham', 90.0);
INSERT INTO students VALUES (2, 1, 'Sakshi', 92.0);
INSERT INTO students VALUES (3, 1, 'Prisha', 94.0);
INSERT INTO students VALUES (1, 2, 'Madhura', 95.0);
INSERT INTO students VALUES (2, 2, 'Om', 96.0);

SELECT * FROM students;

CREATE UNIQUE INDEX idx_students ON students(roll,std);

INSERT INTO students VALUES (3, 1, 'Rachana', 99.0);
-- error: roll 3 in std 1 already exists

SHOW INDEXES FROM students;

DROP INDEX idx_students ON students;

SHOW INDEXES FROM students;
```

Constraints

NOT NULL and UNIQUE

```
DROP TABLE IF EXISTS contacts;

CREATE TABLE contacts(
    name VARCHAR(40) NOT NULL,
    phone CHAR(14) UNIQUE NOT NULL, -- column level constraint
    email VARCHAR(40),
    UNIQUE(email) -- table level constraint (name is auto-generated)
);

INSERT INTO contacts VALUES('Nilesh', '9527331338', 'nilesh@sunbeaminfo.com');
INSERT INTO contacts VALUES(NULL, '9876543210', NULL);
```

```
-- error: name cannot be NULL
INSERT INTO contacts(email,phone) VALUES('james.bond@london.com', '9876543210');
-- error: name cannot be NULL
INSERT INTO contacts VALUES('Abhishek', '9822012345', NULL);
INSERT INTO contacts VALUES('Amitabh', '9822012346', NULL);

INSERT INTO contacts VALUES('Jaya', '9822012345', NULL);
-- error: phone cannot be duplicated
INSERT INTO contacts VALUES('Jaya', NULL, NULL);
-- error: phone cannot be NULL

SELECT * FROM contacts;

DESCRIBE contacts;

SHOW INDEXES FROM contacts;
```

- To delete unique constraint one can delete unique index created with it.
- Constraints can be modified after creating table using ALTER statement.

```
DROP INDEX idx_emo_deptno_job ON emp;

DESCRIBE emp;

ALTER TABLE emp MODIFY ename VARCHAR(40) NOT NULL;

DESCRIBE emp;

INSERT INTO emp(empno, sal) VALUES(1000, 3000.0);
-- error: ename cannot be NULL

ALTER TABLE emp ADD CONSTRAINT unique_empno UNIQUE(empno);

DESCRIBE emp;

INSERT INTO emp(empno, ename) VALUES(7900, 'JOHN');
-- error: ename cannot be duplicated

ALTER TABLE emp DROP CONSTRAINT unique_empno;

DESCRIBE emp;
```

Primary Key

```
-- primary key
CREATE TABLE customers(
    name VARCHAR(40),
    email VARCHAR(30) PRIMARY KEY, -- column level
    phone CHAR(14),
```

```
    password VARCHAR(20),
    addr VARCHAR(80),
    birth DATE
);

-- composite primary key
CREATE TABLE students(
    std INT,
    roll INT,
    name VARCHAR(40),
    marks DECIMAL(5,2),
    PRIMARY KEY (std, roll) -- must be on table level
);

CREATE TABLE students(
    std INT PRIMARY KEY,
    roll INT PRIMARY KEY,
    name VARCHAR(40),
    marks DECIMAL(5,2)
);
-- error: only one primary key is allowed

-- surrogate primary key
CREATE TABLE products(
    id INT PRIMARY KEY,
    name VARCHAR(40),
    category CHAR(20),
    price DECIMAL(7,2),
    quantity INT,
    rating DECIMAL(2,1)
);
-- id is extra column added into table to use as primary key -- surrogate primary
key
-- mostly surrogate PK are auto-generated (serial number)

CREATE TABLE products(
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(40),
    category CHAR(20),
    price DECIMAL(7,2),
    quantity INT,
    rating DECIMAL(2,1)
);

DESCRIBE products;

INSERT INTO products(name,price) VALUES('Notebook', 40);
INSERT INTO products(name,price) VALUES('Pencil', 4);
INSERT INTO products(name,price) VALUES('File', 25);

SELECT * FROM products;

ALTER TABLE products AUTO_INCREMENT = 100;
```

```
INSERT INTO products(name,price) VALUES('Pen', 10);
INSERT INTO products(name,price) VALUES('Eraser', 5);

SELECT * FROM products;
```

- Oracle have a concept of SEQUENCE for auto-generated values.

Foreign Key

```
INSERT INTO emp(empno, ename, deptno) VALUES (1000, 'Motu', 50);
-- allowed because no FK constraint present. due to this Motu is orphan record.

DELETE FROM emp WHERE empno = 1000;

SELECT * FROM emp;

ALTER TABLE dept ADD PRIMARY KEY (deptno);

DESCRIBE dept;

SHOW INDEXES FROM dept;

ALTER TABLE emp ADD FOREIGN KEY(deptno) REFERENCES dept(deptno);

DESCRIBE emp;

SHOW INDEXES FROM emp;

SELECT e.ename, d.dname FROM emp e
JOIN dept d ON e.deptno = d.deptno;

EXPLAIN FORMAT=JSON
SELECT e.ename, d.dname FROM emp e
JOIN dept d ON e.deptno = d.deptno;

INSERT INTO emp(empno, ename, deptno) VALUES (1000, 'Motu', 50);
-- error: PK is not present in dept
```

```
DROP TABLE IF EXISTS depts;
DROP TABLE IF EXISTS emps;

CREATE TABLE depts(deptno INT, dname VARCHAR(20), PRIMARY KEY(deptno));

CREATE TABLE emps(empno INT, ename VARCHAR(20), deptno INT, mgr INT,
FOREIGN KEY (deptno) REFERENCES depts(deptno));
```

```
DROP TABLE IF EXISTS depts;
DROP TABLE IF EXISTS emps;

CREATE TABLE depts(deptno INT, dname VARCHAR(20), PRIMARY KEY(deptno));

CREATE TABLE emps(empno INT, ename VARCHAR(20), deptno INT, mgr INT,
PRIMARY KEY (empno),
FOREIGN KEY (deptno) REFERENCES depts(deptno),
FOREIGN KEY (mgr) REFERENCES emps(empno));

-- parent rows should be inserted first
INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

-- emps must be in a order to handle self-referencing fk.
INSERT INTO emps VALUES (5, 'Sarang', 40, NULL);
INSERT INTO emps VALUES (4, 'Nitin', 40, 5);
INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);
```

```
DROP TABLE IF EXISTS emps;
DROP TABLE IF EXISTS depts;

CREATE TABLE depts(deptno INT, dname VARCHAR(20), PRIMARY KEY(deptno));

CREATE TABLE emps(empno INT, ename VARCHAR(20), deptno INT, mgr INT,
PRIMARY KEY (empno),
FOREIGN KEY (deptno) REFERENCES depts(deptno),
FOREIGN KEY (mgr) REFERENCES emps(empno));

-- disable fk checks temporarily so that data insertion can be faster.
SELECT @@foreign_key_checks;
SET @@foreign_key_checks = 0;
SELECT @@foreign_key_checks;

INSERT INTO emps VALUES (4, 'Nitin', 40, 5);
INSERT INTO emps VALUES (5, 'Sarang', 40, NULL);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);
INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);

INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

SELECT * FROM depts;
SELECT * FROM emps;
```

```
-- enable fk checks after data insertion, so that future inserts/updates will be
validated.
SET @@foreign_key_checks = 1;
SELECT @@foreign_key_checks;

INSERT INTO emps VALUES (6, 'Patlu', 70, 5);
-- error: fk check
```

```
-- Cannot delete parent rows until all child rows are deleted (because it will
make child rows orphan).
DELETE FROM depts WHERE deptno = 40;
-- error
DELETE FROM depts WHERE deptno = 30;
-- allowed: no child rows in emp table for deptno=30

DROP TABLE depts;
-- error: not allowed because all rows in emp will become orphan.

UPDATE depts SET deptno = 60 WHERE deptno = 10;
-- error: not allowed because child rows in emp for dept=10 will become orphan.
```

```
DROP TABLE IF EXISTS emps;
DROP TABLE IF EXISTS depts;

CREATE TABLE depts(deptno INT, dname VARCHAR(20), PRIMARY KEY(deptno));

CREATE TABLE emps(empno INT, ename VARCHAR(20), deptno INT, mgr INT,
PRIMARY KEY (empno),
FOREIGN KEY (deptno) REFERENCES depts(deptno) ON UPDATE CASCADE ON DELETE CASCADE
);
-- this feature is mysql rdbms only.

SET @@foreign_key_checks = 0;

INSERT INTO emps VALUES (4, 'Nitin', 40, 5);
INSERT INTO emps VALUES (5, 'Sarang', 40, NULL);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);
INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);

INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

SET @@foreign_key_checks = 1;

SELECT * FROM depts;
```

```
SELECT * FROM emps;

UPDATE depts SET deptno = 60 WHERE deptno = 10;
-- ON UPDATE CASCADE (in CREATE TABLE emps)
-- when parent is updated 10 --> 60, cascade child updatation 10 --> 60

SELECT * FROM depts;
SELECT * FROM emps;

DELETE FROM depts WHERE deptno = 40;
-- ON DELETE CASCADE (in CREATE TABLE emps)
-- where  parent (40) is deleted, cascade child deletion (emps with dept 40)

SELECT * FROM depts;
SELECT * FROM emps;
```

CHECK

```
CREATE TABLE checked_emp(
empno INT(4),
ename VARCHAR(40),
job VARCHAR(40) CHECK (BINARY job = BINARY UPPER(job)),
mgr INT(4),
hire DATE,
sal DECIMAL(8,2) CHECK (sal BETWEEN 500 AND 5000),
comm DECIMAL(8,2),
deptno INT(4),
CHECK (sal + IFNULL(comm,0.0) >= 800)
);

INSERT INTO checked_emp(empno, ename, job, sal, comm) VALUES (1001, 'JOHN',
'ANALYST', 2900.0, NULL);

INSERT INTO checked_emp(empno, ename, job, sal, comm) VALUES (1002, 'MOTU',
'Director', 4000.0, NULL);
-- error: job must be in upper case

INSERT INTO checked_emp(empno, ename, job, sal, comm) VALUES (1003, 'PATLU',
'SWEEPER', 400.0, NULL);
-- error: sal between 500 and 5000

INSERT INTO checked_emp(empno, ename, job, sal, comm) VALUES (1003, 'PATLU',
'SWEEPER', 800.0, -200.0);
-- error: sal + comm must >= 800
```

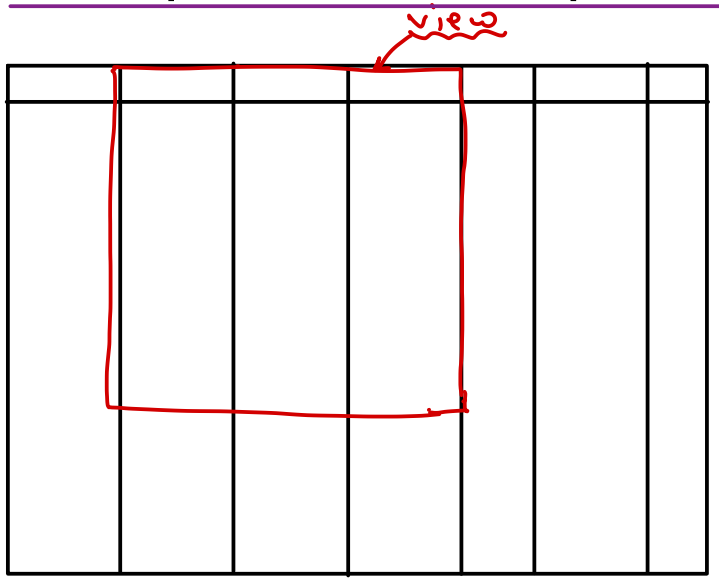
MySQL - RDBMS

Trainer: Mr. Nilesh Ghule

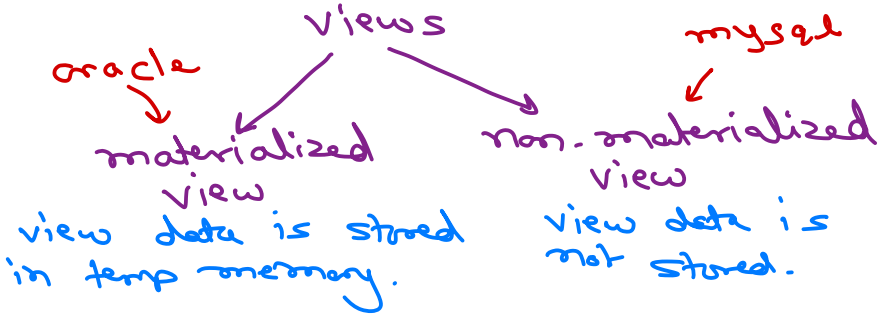
Views

- 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.

→ each query on view will 'internally execute select query first. & process on top of it.




```
Create view view_name
AS SELECT ..... ;
```



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 view affects underlying table.
- View can be created with CHECK OPTION to ensure that DML operations can be performed only the data visible in that view.

View

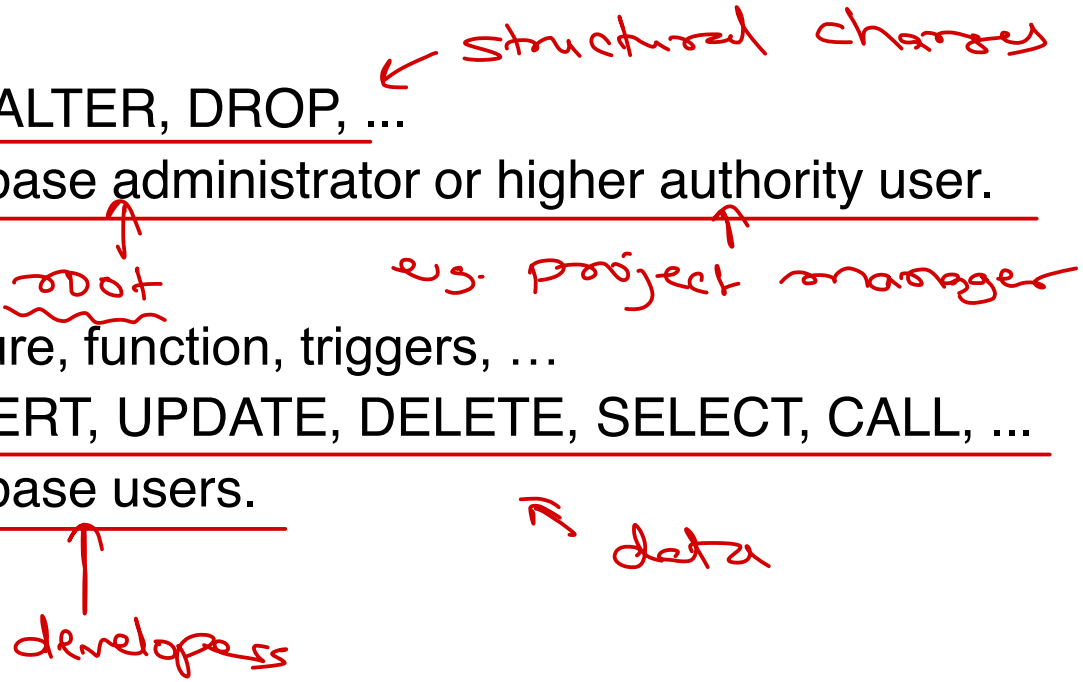
- Views can be differentiated with: SHOW FULL TABLES.
- Views can be dropped with DROP VIEW statement.
- View can be based on another view.

create view view2 as
select * from view1 where ---;

- Applications of views
  - Security: Providing limited access to the data.
  - Hide source code of the table.
  - Simplifies complex queries.

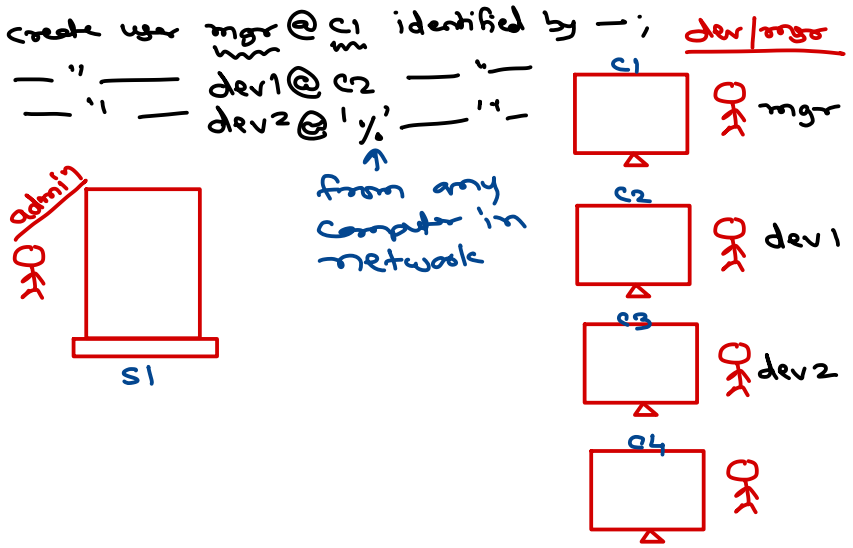
Data Control Language

- 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, ALTER, DROP, ...
  - Typically these privileges are given to the database administrator or higher authority user.
- 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.



User Management

- User management is responsibility of admin (root).
- New user can be created using CREATE USER.
  - CREATE USER user@host IDENTIFIED BY 'password';
  - host can be hostname of server, localhost (current system) or '%' for all client systems.
- Permissions for the user can be listed using SHOW GRANTS command.
  - SHOW GRANTS FOR user@host;
- Users can be deleted using DROP USER.
  - DROP USER user@host;
- Change user password.
  - ALTER USER user@host IDENTIFIED BY 'new_password';
  - FLUSH PRIVILEGES;



Data Control Language

- Permissions are given to user using GRANT command.
  - GRANT CREATE ON db.* TO user@host;
  - GRANT CREATE ON *.* 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 assigned to any user can be withdrawn using REVOKE command.
  - REVOKE SELECT, INSERT ON db.table FROM user@host;
- Permissions can be activated by FLUSH PRIVILEGES.
  - System GRANT tables are reloaded by this command. Auto done after GRANT, REVOKE.
  - Command is necessary if GRANT tables are modified using DML operations.

DDL – ALTER statement

- 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;

PSM - Agenda - Persistent Storage Module.

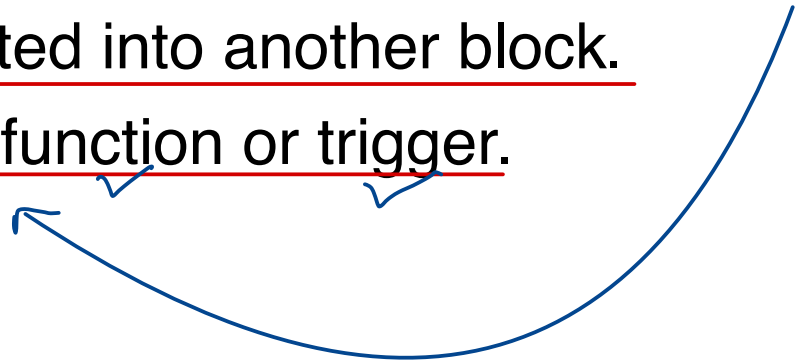
- MySQL Programming
- Stored procedure
- Exceptions
- Function
- Trigger

program → set of instructions

MySQL program → set of sql stmts.
along with programming constructs e.g.
loops, if-else, ...

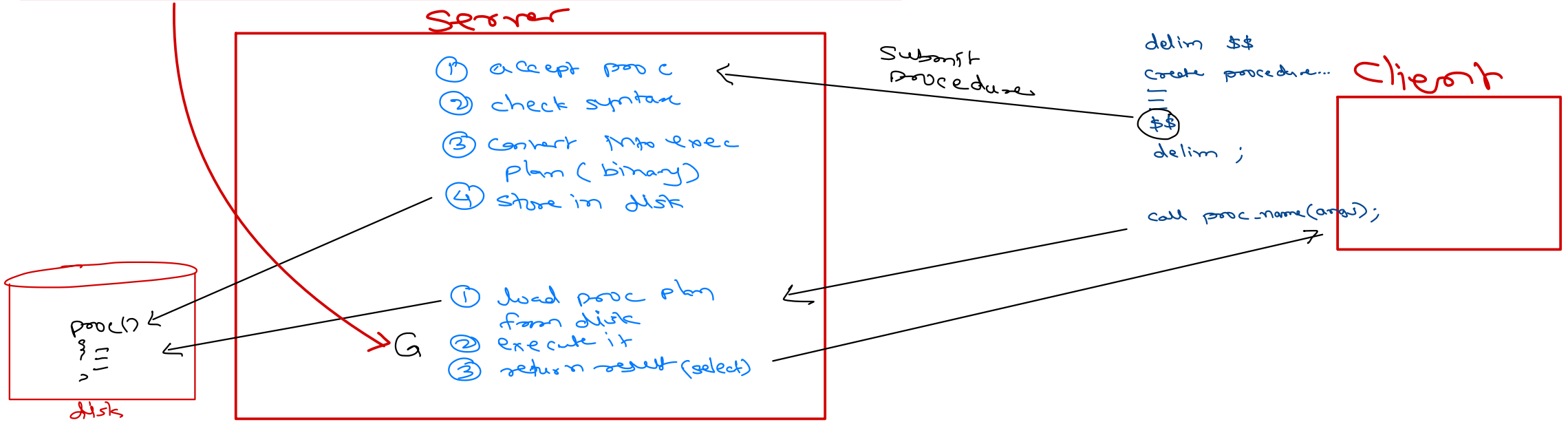
MySQL Programming

- 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;
```

```
-- 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 ;


SOURCE /path/to/01_hello.sql

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!

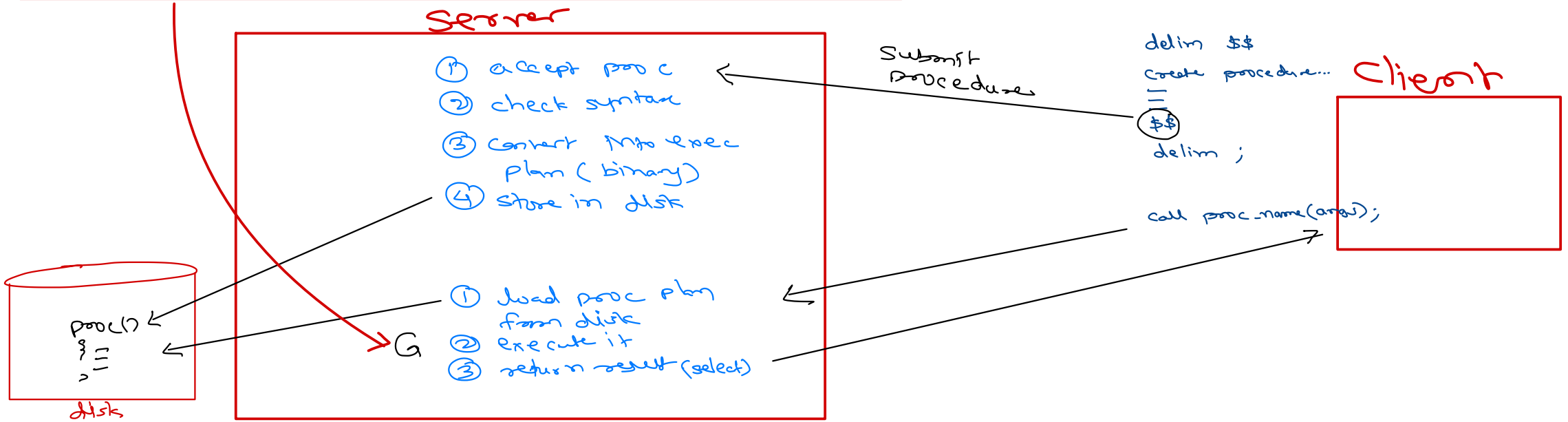
Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Trainer: Mr. Nilesh Ghule

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~~ procedure.

like for definition

like for call.

→ eg. results table

→ only last select stmt output will be displayed.

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;
```

← see the result.

```
-- 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 ;

SOURCE /path/to/01_hello.sql

CALL sp_hello();
```

forward slash

no space in whole path.

output will be displayed here. (CLI).

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

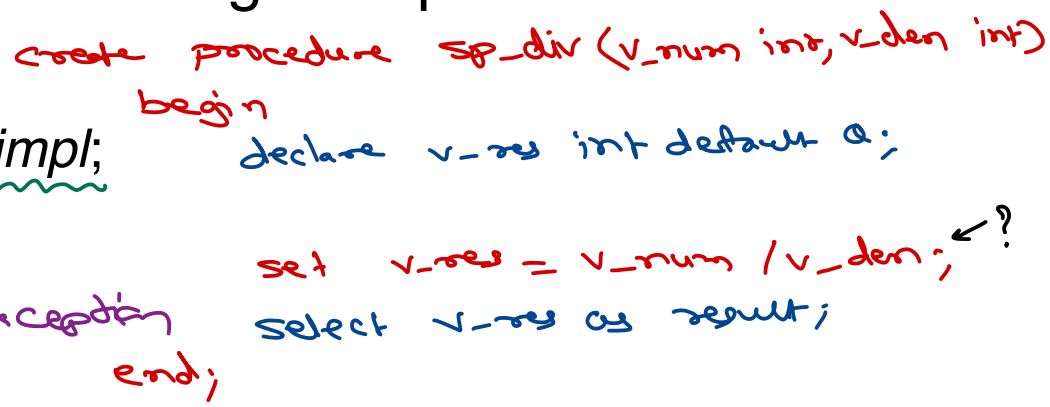
```
SHOW PROCEDURE STATUS
LIKE ‘sp_name’;

SHOW CREATE PROCEDURE sp_name;
```

DROP PROCEDURE

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

MySQL Exceptions / Error Handling

- 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;



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Trainer: Mr. Nilesh Ghule

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 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;
```

MySQL Triggers

- Triggers are supported by all standard RDBMS like Oracle, MySQL, etc.
- Triggers are not supported by WEAK RDBMS like MS-Access.
- 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;
```

SHOW TRIGGERS

```
SHOW TRIGGERS FROM db_name;
```

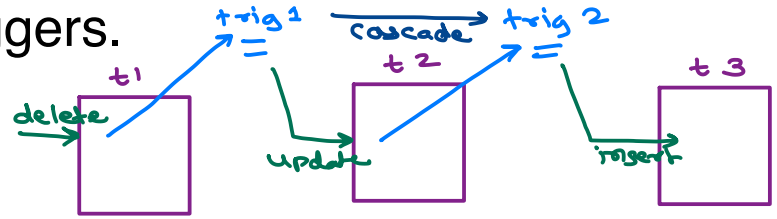
DROP TRIGGER

```
DROP TRIGGER trig_name;
```

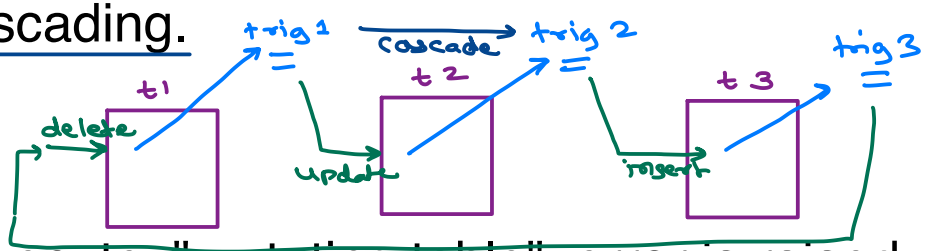
MySQL Triggers

- 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.

backup



- 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.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Agenda

- Cursors
- Functions
- Triggers
- Mongo

Cursor

- T1 (C1) --> 1, 2, 3, 4
- T2 (C2) --> 10, 22, 35, 46

```
DECLARE v_cur1 CURSOR FOR SELECT c1 FROM t1;
DECLARE v_cur2 CURSOR FOR SELECT c2 FROM t2;

OPEN v_cur1;
OPEN v_cur2;

SET v_i = 1;

again: LOOP
    FETCH v_cur1 INTO v1;
    IF v_flag = 1 THEN
        LEAVE again;
    END IF;

    FETCH v_cur2 INTO v2;
    IF v_flag = 1 THEN
        LEAVE again;
    END IF;

    INSERT INTO result VALUES (v_i, CONCAT(v1, ' - ', v2));
    SET v_i = v_i + 1;
END LOOP;

CLOSE v_cur1;
CLOSE v_cur2;
```


Characteristics of MySQL Cursors

- Readonly
  - We can use cursor only for reading from the table.
  - Cannot update or delete from the cursor.
    - SET v_cur1 = (1, 'NewName'); -- not allowed
  - To update or delete programmer can use UPDATE/DELETE queries.

- Non-scrollable
  - Cursor is forward only.
  - Reverse traversal or random access of rows is not supported.
  - When FETCH is done, current row is accessed and cursor automatically go to next row.
  - We can close cursor and reopen it. Now it again start iterating from the start.
- Asensitive
  - When cursor is opened, the addresses of all rows (as per SELECT query) are recorded into the cursor (internally). These rows are accessed one by one (using FETCH).
  - While cursor is in use, if other client modify any of the rows, then cursor get modified values. Because cursor is only having address of rows.
  - Cursor is not creating copy of the rows. Hence MySQL cursors are faster.

Functions

- MySQL Function Types
  - DETERMINISTIC
    - If input is same, output will remain same ALWAYS.
    - Internally MySQL cache input values and corresponding output.
    - If same input is given again, directly output may return to speedup execution.
  - NOT DETERMINISTIC
    - Even if input is same, output may differ.
    - Output also depend on current date-time or state of table or database settings.
    - These functions cannot be speedup.

Triggers

```
DROP TABLE IF EXISTS accounts;

CREATE TABLE accounts(id INT, type CHAR(20), balance DOUBLE);
INSERT INTO accounts VALUES (1, 'Saving', 10000);
INSERT INTO accounts VALUES (2, 'Saving', 2000);
INSERT INTO accounts VALUES (3, 'Current', 25000);
INSERT INTO accounts VALUES (4, 'Saving', 7000);

CREATE TABLE transactions(accid INT, type CHAR(20), tim DATETIME, amount DOUBLE);
```

- Implement trigger -- psm19.sql

```
SELECT * FROM accounts;

INSERT INTO transactions VALUES (1, 'WITHDRAW', NOW(), 2000);

SELECT * FROM accounts;

INSERT INTO transactions VALUES (2, 'DEPOSIT', NOW(), 500);

SELECT * FROM accounts;
```

```
SELECT * FROM transactions;
```

```
SHOW TRIGGERS FROM classwork;
```

- Assign: When sal of emp is modified, make entry into result table of old and new sal.
    - Trigger --> BEFORE UPDATE ON emp

```
CREATE TRIGGER trig_salchange
AFTER UPDATE ON emp
FOR EACH ROW
BEGIN
    DECLARE v_empno DOUBLE DEFAULT OLD.empno;
    DECLARE v_oldsal DOUBLE DEFAULT OLD.sal;
    DECLARE v_newsal DOUBLE DEFAULT NEW.sal;
    IF OLD.sal != NEW.sal THEN
        INSERT INTO result VALUES (v_empno, CONCAT(v_oldsal, ' --> ', v_newsal));
    END IF;
END;
$$
```

```
START TRANSACTION;

UPDATE emp SET sal = 1200 WHERE empno = 7900;
-- OLD.sal = 950.0 --> NEW.sal = 1200.0
-- OLD.empno = NEW.empno -- both are same (because we are not modifying empno.)

COMMIT; -- both changes will be permanent
-- or
ROLLBACK; -- both changes will be discarded
```

NoSQL - MongoDb

- Mongo Server is running in backgroud (mongod).
    - Version: 3.6+
- Open command prompt -- Start Mongo Shell.
- terminal> mongo
    - By default security is disabled in Mongo.
- Try next queries in mongo shell ">".
- JS is case sensitive. Most of Mongo queries are case sensitive.

```

```

```
show databases;

use classwork;
// classwork db will be created when first record is added in it.

db;
// db is keyword -- represent current database.

show collections;

db.people.insert({
    name: "Nilesh",
    age: 37,
    addr: {
        city: "Pune",
        pin: 411037
    },
    email: "nilesh@sunbeaminfo.com"
});
// insert a document {...} json into 'people' collection in current database (db).

show collections;

show databases;

db.people.insert({
    name: "Nitin",
    mobile: "9881208115",
    email: "nitin@sunbeaminfo.com"
});

db.people.insertMany([
{
    name: "Prashant",
    mobile: "9881208114",
    email: "prashant@sunbeaminfo.com"
},
{
    name: "Sunbeam Infotech",
    phone: "020-24260308",
    website: "www.sunbeaminfo.com"
}
]);
```

```
db.people.find();

db.people.insert({
    _id: 1,
    name: "Sarang",
    addr: {
        area: "Karad",
```

```
        city: "Satara"
    }
});

db.people.find();

db.people.insert({
    _id: 1,
    name: "Rachana",
    addr: {
        area: "Karad",
        city: "Satara"
    }
});
// error: _id cannot be duplicated

db.people.insert({
    _id: 2,
    name: "Rachana",
    addr: {
        area: "Karad",
        city: "Satara"
    }
});

db.people.find();
```

- find() operation returns mongo cursor.
- Cursor is used to access elements one by one.
- Cursor functions:
    - pretty() -- format the output
    - limit(n) -- LIMIT n
    - skip(m) -- skip m records
    - sort() -- ORDER BY

```
db.people.find().pretty();

// LIMIT 2;
db.people.find().limit(2);

db.people.find().skip(2);

// LIMIT 2, 3;
db.people.find().skip(2).limit(3);

db.people.find().skip(2).limit(3).pretty();
```

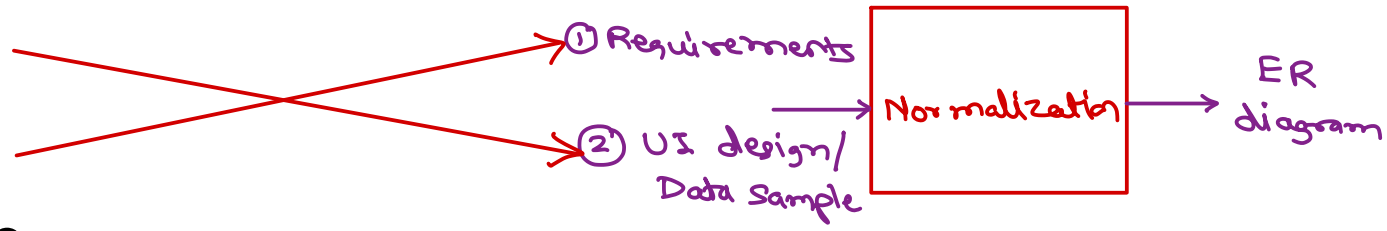
```
db.dept.remove({});
db.emp.remove({});

db.dept.insert({_id:10,dname:"ACCOUNTING",loc:"NEW YORK"});
db.dept.insert({_id:20,dname:"RESEARCH",loc:"DALLAS"});
db.dept.insert({_id:30,dname:"SALES",loc:"CHICAGO"});
db.dept.insert({_id:40,dname:"OPERATIONS",loc:"BOSTON"});

db.emp.insert({_id:7369,ename:"SMITH",job:"CLERK",mgr:7902,sal:800.00,deptno:20});
db.emp.insert({_id:7499,ename:"ALLEN",job:"SALESMAN",mgr:7698,sal:1600.00,comm:300
.00,deptno:30});
db.emp.insert({_id:7521,ename:"WARD",job:"SALESMAN",mgr:7698,sal:1250.00,comm:500.
00,deptno:30});
db.emp.insert({_id:7566,ename:"JONES",job:"MANAGER",mgr:7839,sal:2975.00,deptno:20
});
db.emp.insert({_id:7654,ename:"MARTIN",job:"SALESMAN",mgr:7698,sal:1250.00,comm:14
00.00,deptno:30});
db.emp.insert({_id:7698,ename:"BLAKE",job:"MANAGER",mgr:7839,sal:2850.00,deptno:30
});
db.emp.insert({_id:7782,ename:"CLARK",job:"MANAGER",mgr:7839,sal:2450.00,deptno:10
});
db.emp.insert({_id:7788,ename:"SCOTT",job:"ANALYST",mgr:7566,sal:3000.00,deptno:20
});
db.emp.insert({_id:7839,ename:"KING",job:"PRESIDENT",sal:5000.00,deptno:10});
db.emp.insert({_id:7844,ename:"TURNER",job:"SALESMAN",mgr:7698,sal:1500.00,comm:0.
00,deptno:30});
db.emp.insert({_id:7876,ename:"ADAMS",job:"CLERK",mgr:7788,sal:1100.00,deptno:20})
;
db.emp.insert({_id:7900,ename:"JAMES",job:"CLERK",mgr:7698,sal:950.00,deptno:30});
db.emp.insert({_id:7902,ename:"FORD",job:"ANALYST",mgr:7566,sal:3000.00,deptno:20}
);
db.emp.insert({_id:7934,ename:"MILLER",job:"CLERK",mgr:7782,sal:1300.00,deptno:10}
);

db.emp.find();

// ORDER BY sal ASC
db.emp.find().sort({sal: 1});

// ORDER BY sal DESC
db.emp.find().sort({sal: -1});

// ORDER BY deptno ASC, sal DESC
db.emp.find().sort({deptno: 1, sal: -1});

// ORDER BY sal DESC LIMIT 1;
db.emp.find().sort({sal: -1}).limit(1).pretty();
```

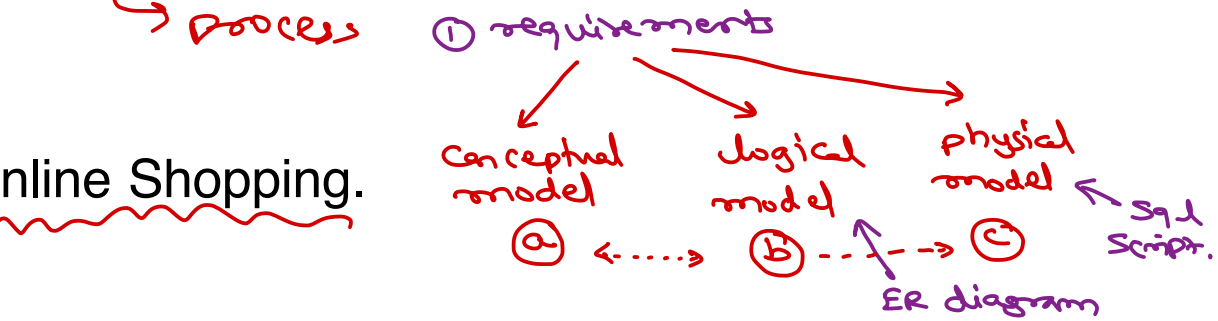
MySQL - RDBMS

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
  - Update anomaly
  - Delete anomaly

Normalization

- 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

- 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.

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.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

NoSQL - Mongo

Agenda

- Mongo CRUD
- NoSQL concepts
- Normalization
- Codd's rules

Mongo

Insert

```
db.dept.insert({
    _id: 50,
    dname: "Training",
    loc: "Pune"
});
// insert record in dept collection
// if dept collection doesn't exists, it will be auto-created
```

```
// can also create collection  explicitly
db.createCollection("students");
```

- If _id field is not given, id is auto-generated by the "client".
- "unique" auto-generated id is of 12 bytes.
    - client process id (2 bytes)
    - client machine id (3 bytes) (each client get unique id from server upon connection)
    - timestamp (4 bytes) -- time at which record is inserted
    - counter (3 bytes) -- counter managed by client

Query

- Like SELECT.
- db.col.find({criteria}, {projection});
    - criteria -- WHERE clause
    - selected fields
- Projection
    - 1: display field, 0: hide field
    - cannot combine display & hide.
        - { _id: 1, ename: 1, sal: 1 }
        - { job: 0, sal: 0, hire: 0, comm: 0 }
        - { ename: 1, sal: 0, hire: 1, comm: 0 } --> error
    - By default id is visible.

- Criteria
  - Relational: $lt, $gt, $lte, $gte, $eq, $ne
  - Logical: $and, $or, $nor
  - Other Operators: $in, $nin, $regex, $type, ...

```
use classwork;

show collections;
```

```
db.emp.find();

db.emp.find({}, { _id: 1, ename: 1, sal: 1 });

db.emp.find({}, { job: 0, sal: 0, hire: 0, comm: 0 });

db.emp.find({}, { ename: 1, sal: 0, hire: 1, comm: 0 }); // error

db.emp.find({}, { ename: 1, job: 1, sal: 1 });
// id is default included

db.emp.find({}, { _id: 0, ename: 1, job: 1, sal: 1 });
// hide id -- exception -- hiding id is allowed in inclusion projection.
```

```
// empty criteria -- all records
db.emp.find({});

// find emp with sal > 2500
db.emp.find({ sal: { $gt: 2500 } });

// find clerk
db.emp.find({ job: { $eq: 'CLERK' } });
db.emp.find({ job: 'CLERK' });

// find all emps working in dept 10 and 20.
db.emp.find({ deptno: { $in: [10, 20] } });

// find all CLERK working in dept 20.
db.emp.find({ $and: [
    { job: 'CLERK' },
    { deptno: 20 }
] });

// find emps with sal <= 1500 or job ANALYST.
db.emp.find({ $or: [
    { sal: { $lte: 5000 } },
    { job: 'ANALYST' }
]});
```

```
// find all emps whose name start with S.
db.emp.find({
    ename: { $regex: /^S/ }
});

db.emp.find({
    ename: { $regex: /^s/i }
});
// --> /regex/mode
// regex -- using wild card chars
// mode -- "i": case insensitive
```

Update/Upsert

- db.col.update({criteria}, {new_object or changes using $set});
  - by default upsert = false.
- updateOne(), updateMany()
- UPSERT operation --> If exists then UPDATE, otherwise INSERT.
  - db.col.update({criteria}, {new_object or changes using $set}, true);
  - arg 3: upsert = true
  - If record doesn's exists, a new record is created with given criteria and then it is updated with given values.

```
db.dept.insert({_id: 50, dname: 'TRAINING', loc: 'PUNE'});

db.dept.find();

db.dept.update({_id: 50}, { loc: 'BANGLORE' });
// {_id: 50, dname: 'TRAINING', loc: 'PUNE'} replaced by {_id: 50, loc: 'BANGLORE'
}

db.dept.find();

db.emp.update({ ename: 'KING' }, { sal: 5500 });
// whole record is replaced by { _id + sal }

db.emp.find();

db.emp.update({ ename: 'JAMES' }, {
    $set: {
        sal: 1000
    }
});

db.emp.find();

db.emp.update({ ename: 'JAMES' }, {
    $set: {
        sal: 1100,
        mgr: 7836
```

```
    }
});

// change sal=1200, comm=200, job=SALESMAN for emp with name = 'JOHN'
db.emp.update( {ename: 'JOHN'}, {
    $set: {
        sal: 1200,
        comm: 200,
        job: 'SALESMAN'
    }
});
// no records found, and hence no records modified

db.emp.update( {ename: 'JOHN'}, {
    $set: {
        sal: 1200,
        comm: 200,
        job: 'SALESMAN'
    }
},
true );

db.emp.find();

db.dept.update({_id: 60}, { $set: { dname: "SECURITY" } }, true);

db.dept.find().count();
```

Delete

- db.col.remove({criteria}); --> delete one or more records as per criteria
- db.col.deleteOne({criteria}); --> delete first as per criteria
- db.col.deleteMany({criteria}); --> delete one or more records as per criteria

```
db.dept.find();

db.dept.deleteMany({ _id: { $gte : 50 } });

db.dept.find();

db.dept.deleteMany({});
// delete all records (but not collection)

show collections;

db.dept.drop();

show collections;
```

Aggregation Pipeline

- Not in syllabus
- Group By, Projection, Joins, ...
- Pipeline -- set of stages

```
db.collection_name.aggregate([
{
    $group: { ... }
},
{
    $match: { ... }
},
{
    $sort: { ... }
},
{
    $limit: { ... }
}
]);
```

Indexes

- faster searching
- Types of Indexes
    - Simple index
    - Unique index
        - By default _id is unique index.
    - Composite index
    - TTL index (Time To Live)
        - Auto expire/delete object after certain time.
    - GeoSpatial index
        - Location based (long + lat) analysis

```
db.emp.createIndex({
    'job': 1
});

db.emp.createIndex({
    'ename': 1,
}, {
    unique: true
});

db.emp.createIndex({
    'deptno': 1,
    'job': 1
});

db.emp.getIndexes();
```

Mongo Import

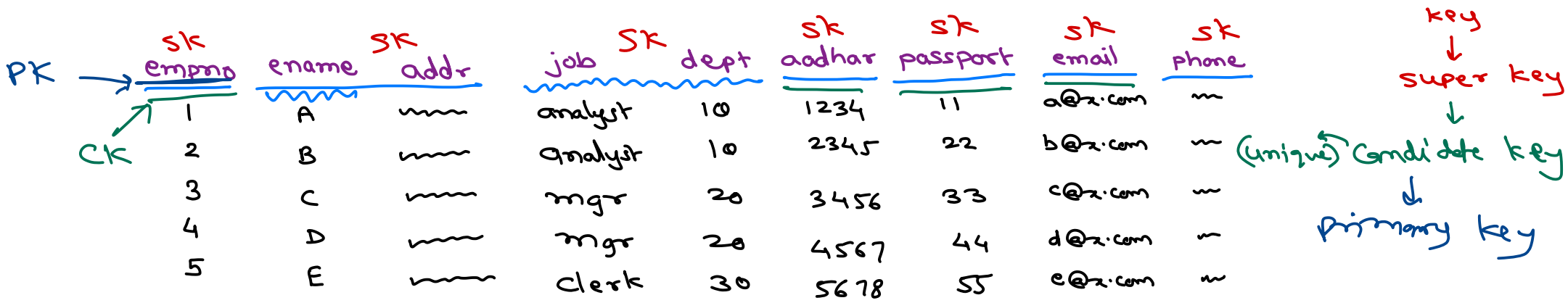
- Import form JS script.
    - cmd> mongo -d classwork empdept.js
- MongoImport
    - cmd> mongoimport -type csv -headerline -d classwork -c emp emp.csv

MySQL - RDBMS

Trainer: Mr. Nilesh Ghule

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 (designer) 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.



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. → eg. adding computed columns, ...
- 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.

maths
↓
Set theory

- 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

- 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.

- 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.

Codd's rules

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

→ simple views

- 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 load →

- 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.

- Rule 9: Logical data independence: → use views
  - 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.

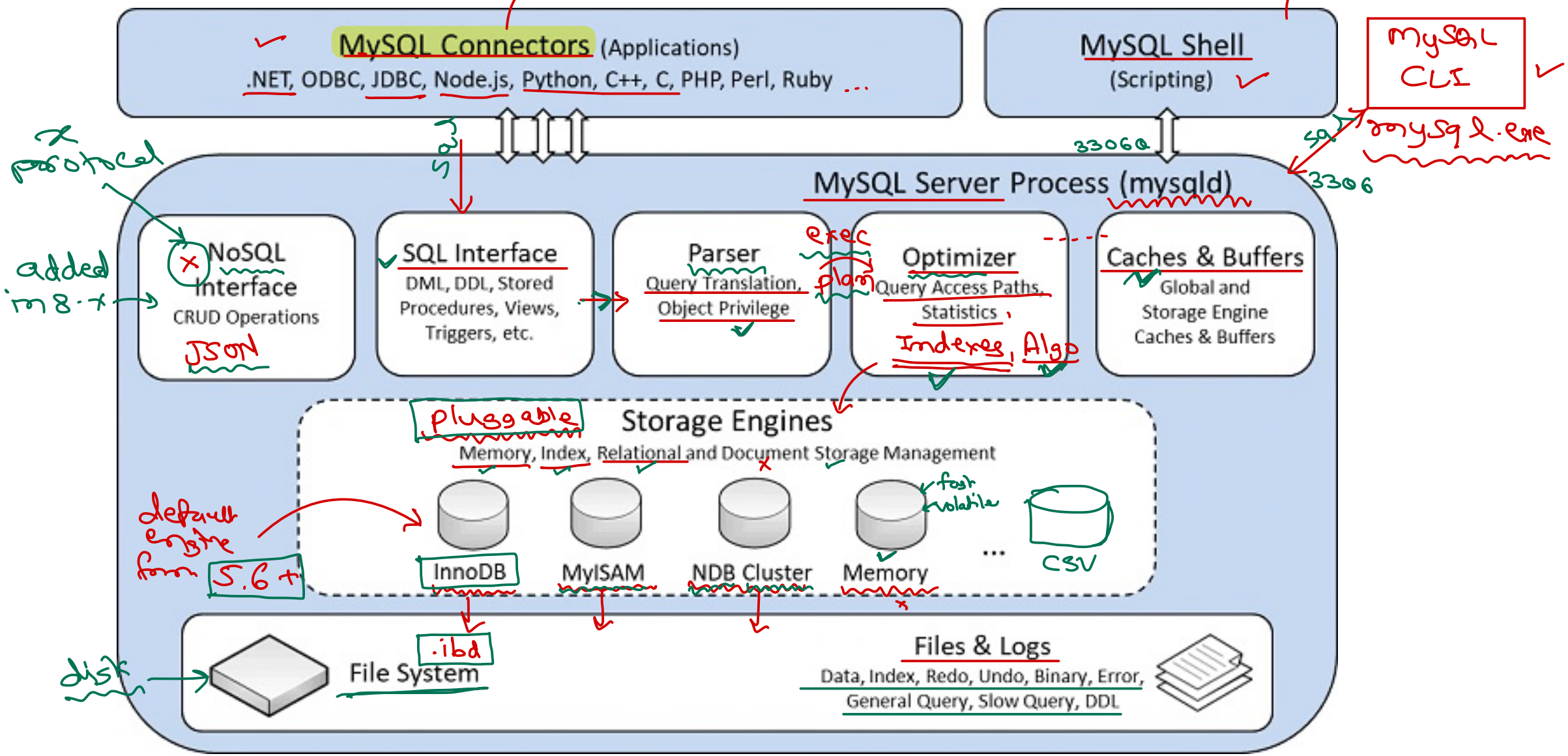
Codd's rules

- 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.

- 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).

MySQL Architecture



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

MySQL - RDBMS

Agenda

- Normalization
- Codd's rules
- MySQL Architecture
- RDBMS to NoSQL
- Full Text Searches
- Temporary Tables

Normalization

ER Diagram

- Lucid Chart (for ED diagram)
- Plant UML (for ED diagram)
    - https://www.planttext.com/
    - Syntax: https://plantuml.com/

```
@startuml

entity customers {
    - cid
    - cname
    - caddr
    - cstreet
    - ccity
    - cpin
    - cphone
}

entity orders {
    - oid
    - odate
    - odeldate
}

entity products {
    - pid
    - pname
    - prate
}

entity order_details {
    - oid
    - pid
    - pqty
}
```

```
entity city_pin {
    - pin
    - city
}

city_pin "1" -left- "*" customers
customers "1" -left- "*" orders
orders "1" -left- "*" order_details
products "1" -right- "*" order_details

@enduml
```

Physical Model

- SQL queries to create tables, relations and constraints.

Using project database

- CRUD operations on the tables
- Advanced reports -- Analysis

Analysis

- Which product is sold maximum in last year?
    - GROUP BY + JOIN (products & order_details)
- Find top 10 customers (max amount orders).
    - GROUP BY + JOIN (products, order_details, orders, customers)
    - If orders table also keep "final_bill" then JOIN (orders, customers)
- Find top 2 cities making maximum business.
    - GROUP BY + JOIN (products, order_details, orders, customers, city_pin)
    - If orders table also keep "final_bill" then JOIN (orders, customers, city_pin)

MySQL Architecture

```
CREATE TABLE items(
    id INT PRIMARY KEY,
    name VARCHAR(20),
    price DOUBLE
) ENGINE=MyISAM;

INSERT INTO items VALUES(1, 'Item1', 200.00);
INSERT INTO items VALUES(2, 'Item2', 100.00);

CREATE TABLE cust(
    id INT NOT NULL,
    name VARCHAR(20) NOT NULL
) ENGINE=CSV;

INSERT INTO cust VALUES (1, 'Nilesh'), (2, 'Nitin'), (3, 'Yogesh'), (4, 'Vijay'),
```

```
(5, 'Soniya');

SELECT * FROM cust;
```

Physical architecture

- Configuration files
  - /etc/mysql/my.cnf
- Installed Files
  - Executable files
    - mysqld
    - mysqladmin
    - mysql
    - mysqldump
    - ...
  - Log files
    - pid files
    - socket files
    - document files
    - libraries
- Data Files
  - Data directory
    - Server logs
    - Status files
    - Innodb tablespaces & log buffer
  - Database directory files
    - Data & Index files (.ibd)
    - Object structure files (.frm, .opt)

Logical architecture

- Refer diagram in slides.
- Client
- Server (mysqld)
  - Accept & process client requests
  - Multi-threaded process
  - Dynamic memory allocation
    - Global allocation
    - Session allocation
- Parser
  - SQL syntax checking.
  - Generate sql_id for query.
  - Check user authentication.
- Optimizer
  - Generate efficient query execution plan
  - Make use of appropriate indexes
  - Check user authorization.

- Query cache
    - Server level (global) cache
    - Speed up execution if identical query is executed previously
- Key cache
    - Cache indexes
    - Only for MyISAM engine
- Storage engine
    - Responsible for storing data into files.
    - Features like transaction, storage size, speed depends on engine.
    - Supports multiple storage engines
    - Can be changed on the fly (table creation)
    - Important engines are InnoDB, MyISAM, NDB, Memory, Archive, CSV.

InnoDB engine

- Fully transactional ACID.
- Table & Row-level locking.
- Offers REDO and UNDO for transactions.
- Shared file to store objects (Data and Index in the same file - .ibd)
- InnoDB Read physical data and build logical structure (Blocks and Rows)
- Logical storage called as TABLESPACE.
- Data storage in tablespace
    - Multiple data files
    - Logical object structure using InnoDB data and log buffer

MyISAM

- Non-transactional storage engine
- Table-level locking
- Speed for read
- Data storage in files and use key, metadata and query cache
    - .frm for table structure
    - .myi for table index
    - .myd for table data

NDB

- Fully Transactional and ACID Storage engine.
- Row-level locking.
- Offers REDO and UNDO for transactions.
- NDB use logical data with own buffer for each NDB engine.
- Clustering: Distribution execution of data and using multiple mysqld.

MySQL as NoSQL

- "Flexible Schema" -- JSON datatype
- Still ACID transactions -- Tight consistency
- Indexing on JSON fields

- Horizontal scaling limited -- GB to TB.

RDBMS to NoSQL migration

- An application is developed using RDBMS & in use.
- For getting advantages of NoSQL, we want to shift to NoSQL databases.
- This includes redevelopment of whole application, specially data layer of the application.
- Migrating from RDBMS to NoSQL will change from (NoSQL) database to database.

Migration from MySQL to MongoDb

1. Understand structure of data in MySQL i.e. understand tables, constraints, relations among tables & indexes.
2. Define data model for MongoDb. Make appropriate use of embedded model & reference model. Define collections, validations & indexes.
3. Create views in RDBMS which will make data available as per requirement of Mongo Db model.
4. Write a custom script that will read data from views & write into JSON files (in desired format).
5. In mongo db create collection, then import all json files using **mongoimport** command.
    - mongoimport -d dacdb -c emp emp.json
6. Create indexes & validators on mongo collections.

Full Text Search

- FTS allows keyword based searches. Need not to match exact text.
- FTS is applicable only for CHAR, VARCHAR and TEXT types.
- FTS is different than LIKE or REGEXP.
- FTS uses Natural Language Processing to make search more user friendly.
- MySQL FTS features
    - High speed - based on fulltext index.
    - Moderate index size - index size is not too large.
    - Dynamic index - index is auto-updated on DML.

```
CREATE TABLE table_name(
    column_list,
    ...,
    FULLTEXT (column1,column2, ...)
);
-- OR
ALTER TABLE table_name
ADD FULLTEXT(column_name1, column_name2, ...);
-- OR
CREATE FULLTEXT INDEX index_name
ON table_name(idx_column_name, ...);
```

```
ALTER TABLE table_name
DROP INDEX index_name;
```

```
CREATE TABLE tutorial (
id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
title VARCHAR(200),
description TEXT,
FULLTEXT(title,description)
) ENGINE=InnoDB;

INSERT INTO tutorial (title,description) VALUES
('SQL Joins','An SQL JOIN clause combines rows from two or more tables. It creates
a set of rows in a temporary table.'),
('SQL Equi Join','SQL EQUI JOIN performs a JOIN against equality or matching
column(s) values of the associated tables. An equal sign (=) is used as comparison
operator in the where clause to refer equality.'),
('SQL Left Join','The SQL LEFT JOIN, joins two tables and fetches rows based on a
condition, which is matching in both the tables and the unmatched rows will also
be available from the table before the JOIN clause.'),
('SQL Cross Join','The SQL CROSS JOIN produces a result set which is the number of
rows in the first table multiplied by the number of rows in the second table, if
no WHERE clause is used along with CROSS JOIN.'),
('SQL Full Outer Join','In SQL the FULL OUTER JOIN combines the results of both
left and right outer joins and returns all (matched or unmatched) rows from the
tables on both sides of the join clause.'),
('SQL Self Join','A self join is a join in which a table is joined with itself
(which is also called Unary relationships), especially when the table has a
FOREIGN KEY which references its own PRIMARY KEY.');

SELECT id, title, LEFT(description, 40) FROM tutorial;

SELECT * FROM tutorial WHERE MATCH(title,description) AGAINST ('left and right
join');
-- columns must be full text indexed

SELECT * FROM tutorial WHERE MATCH(title,description) AGAINST ('left right');

SELECT * FROM tutorial WHERE MATCH(title,description) AGAINST ('left right' IN
NATURAL LANGUAGE MODE);

SELECT id, MATCH(title,description) AGAINST ('left right' IN NATURAL LANGUAGE
MODE) AS score FROM tutorial;
-- relevence depend on the number of words in the row, the number of unique words
in that row, the total number of words in the collection, the number of documents
(rows) that contain a particular word.

SELECT * FROM tutorial WHERE  MATCH(title,description) AGAINST ('+Joins -right' IN
BOOLEAN MODE);

SELECT * FROM tutorial WHERE  MATCH(title,description) AGAINST ('full');

SELECT * FROM tutorial WHERE  MATCH(title,description) AGAINST ('full' WITH QUERY
EXPANSION);
```

FTS Modes

- Natural Language Mode
  - Default mode of searching.
  - Assign relevence to each row based on search terms.
    - 0 - Not similar.
    - +ve - Similar.
- Boolean Mode
  - Expert search based on operators to change relevence.
  -
    - Include, the word must be present.
  - – Exclude, the word must not be present.
  - > Include, and increase ranking value.
  - < Include, and decrease the ranking value.
  - () Group words into subexpressions (allowing them to be included, excluded, ranked, and so forth as a group).
  - ~ Negate a word's ranking value.
  -
    - Wildcard at the end of the word.
  - "" Defines a phrase (as opposed to a list of individual words, the entire phrase is matched for inclusion or exclusion).
- With Query Expansion
  - Search related words as well.

FTS Restrictions

- No spelling mistakes
- Minimum length of the search term defined in MySQL full-text search engine is 4.
- Stop-words are ignored e.g. is, are, and, to, on, in, as, ...

Temporary Tables

- Like Materialized View in Oracle.
- It stores data temporarily in in-memory table -- for current user session.
- Temporarily tables are visible only for current user current session.
- When current session EXIT, temporarily tables are relased.

```
SELECT * FROM emp WHERE sal > 2500;

-- CREATE VIEW v_richemp AS SELECT * FROM emp WHERE sal > 2500;
CREATE TEMPORARY TABLE tmp_richemp AS SELECT * FROM emp WHERE sal > 2500;

SELECT * FROM tmp_richemp;

UPDATE emp SET sal = 5500.0 WHERE ename='KING';
-- change in base table

SELECT * FROM emp;
```

```
SELECT * FROM tmp_richemp;
-- changes are not visible in temp table

SHOW TABLES;

SHOW FULL TABLES;
-- temp tables are not visible

DROP TEMPORARY TABLE tmp_richemp;

SELECT * FROM tmp_richemp;
-- error

CREATE TEMPORARY TABLE tmp_richemp AS SELECT * FROM emp WHERE sal > 2500;
-- new temp table created

SELECT * FROM tmp_richemp;

EXIT;
-- temp tables are destroyed
```

```
SELECT * FROM tmp_richemp;
-- error

SELECT * FROM INNODB_TEMP_TABLE_INFO;
-- see hidden tables in current session (but need PROCESS privilege)
```

Convert this data to 3NF

CustomerName	CustID	Address	SubType	SubAmount	MovieDownloads	DownloadDate
Tom Smith	WS951	5 High Street Aylesbury HP20 4YB	1 Month	£5.00	AV – Avengers Age of Ultron (Sci-Fi)	12/12/2014
Tom Smith	WS951	5 High Street Aylesbury HP20 4YB	1 Month	£5.00	JW – Jurassic World (Sci-Fi)	14/12/2014
Tom Smith	WS951	5 High Street Aylesbury HP20 4YB	1 Month	£5.00	TR – Train (Comedy)	16/12/2014
Tom Smith	WS951	5 High Street Aylesbury HP20 4YB	1 Month	£5.00	IO – Inside Out (Animated)	20/12/2014
Rebecca Zane	AK123	77 Green Street High Wycombe HP14JQ	12 months	£50.00	TG – Terminator Genisys (Sci-Fi)	23/05/2012
Rebecca Zane	AK123	77 Green Street High Wycombe HP14JQ	12 months	£50.00	IO – Inside Out (Animated)	05/11/2013
Rebecca Zane	AK123	77 Green Street High Wycombe HP14JQ	12 months	£50.00	MN – Minions (Animated)	08/01/2015
Rebecca Zane	AK123	77 Green Street High Wycombe HP14JQ	12 months	£50.00	IO – Inside Out (Animated)	08/01/2015

HILLTOP ANIMAL HOSPITAL DATE: JAN 13/2002

INVOICE # 987

MR. RICHARD COOK

123 THIS STREET

MY CITY, ONTARIO

Z5Z 6G6

PET	PROCEDURE		AMOUNT
ROVER	RABIES VACCINATION		30.00
MORRIS	RABIES VACCINATION		24.00
	TOTAL	54.00	
	TAX (8%)	4.32	
	AMOUNT OWING	58.32	

Convert this data to 3NF

INVOICE

INVOICENO	DATE	Name	Address	Pet	PROCEDURE	AMOUNT	TOTAL	TAX (8%)	AMOUNT OWING
987	JAN 13/2002	Mr Richard Cook	123 THIS STREET MY CITY, ONTARIO Z5Z 6G6	ROVER	RABIES VACCINATION	30.00	54.00	4.32	58.32
987	JAN 13/2002	Mr Richard Cook	123 THIS STREET MY CITY, ONTARIO Z5Z 6G6	MORRIS	RABIES VACCINATION	24.00			

INVOICENO	Branch	DATE	Name	Address	Pet	PROCEDURE	AMOUNT	TOTAL	TAX (8%)	AMOUNT OWING
987	HILLTOP ANIMAL HOSPITAL	JAN 13/2002	Mr Richard Cook	123 THIS STREET MY CITY, ONTARIO Z5Z 6G6	ROVER	RABIES VACCINATION	30.00	54.00	4.32	58.32
987	HILLTOP ANIMAL HOSPITAL	JAN 13/2002	Mr Richard Cook	123 THIS STREET MY CITY, ONTARIO Z5Z 6G6	MORRIS	RABIES VACCINATION	24.00			

UNF

UNF

InvoiceNo
Branch
DATE
Name
Address
PetName
PROCEDURE
AMOUNT
TOTAL
TAX (8%)
AMOUNT OWING

bucks new university

For next week convert this data to 3NF

studentid	StudentName	Address	course	modulecode	modulename	tutor	day	room
25000075	Adrian Smith	Beaconsfield	BSc IT Information Technology	CO456	Web	Carlo Lusuardi	Thursday	G5.05
25000075	Adrian Smith	Beaconsfield	BSc IT Information Technology	CO454	Digital Technologies	Hilary Mullen	Wednesday	G5.04
25000075	Adrian Smith	Beaconsfield	BSc IT Information Technology	CO450	Computer Architectures	Justin Luker	Monday	G5.02
25000075	Adrian Smith	Beaconsfield	BSc IT Information Technology	CO457	Business Modelling	Justin Luker	Wednesday	G5.09
25000076	Mohammed Hussain	Milton Keynes	BSc IT Information Technology	CO456	Web	Carlo Lusuardi	Thursday	G5.05
25000076	Mohammed Hussain	Milton Keynes	BSc IT Information Technology	CO454	Digital Technologies	Hilary Mullen	Wednesday	G5.04
25000076	Mohammed Hussain	Milton Keynes	BSc IT Information Technology	CO450	Computer Architectures	Justin Luker	Monday	G5.02
25000076	Mohammed Hussain	Milton Keynes	BSc IT Information Technology	CO457	Business Modelling	Justin Luker	Wednesday	G5.09
25000077	James Miller	Amersham	BSc IT Information Technology	CO456	Web	Carlo Lusuardi	Thursday	G5.05
25000077	James Miller	Amersham	BSc IT Information Technology	CO454	Digital Technologies	Hilary Mullen	Wednesday	G5.04
25000077	James Miller	Amersham	BSc IT Information Technology	CO450	Computer Architectures	Justin Luker	Monday	G5.02
25000077	James Miller	Amersham	BSc IT Information Technology	CO457	Business Modelling	Justin Luker	Wednesday	G5.09
25000078	Jack White	High Wycombe	BSc SFT Software Technologies	CO454	Digital Technologies	Hilary Mullen	Wednesday	G5.04
25000078	Jack White	High Wycombe	BSc SFT Software Technologies	CO450	Computer Architectures	Justin Luker	Monday	G5.02
25000078	Jack White	High Wycombe	BSc SFT Software Technologies	CO452	Programming Concepts	Richard Jones	Tuesday	G5.03
25000078	Jack White	High Wycombe	BSc SFT Software Technologies	CO455	User Experience	Kevin Maher	Tuesday	G5.08
2500079	Michael Cane	Aylesbury	BSc SFT Software Technologies	CO454	Digital Technologies	Hilary Mullen	Wednesday	G5.04
2500079	Michael Cane	Aylesbury	BSc SFT Software Technologies	CO450	Computer Architectures	Justin Luker	Monday	G5.02
2500079	Michael Cane	Aylesbury	BSc SFT Software Technologies	CO452	Programming Concepts	Richard Jones	Tuesday	G5.03
2500079	Michael Cane	Aylesbury	BSc SFT Software Technologies	CO455	User Experience	Kevin Maher	Tuesday	G5.08
25000080	Joe Bloggs	Amersham	BSc SFT Software Technologies	CO454	Digital Technologies	Hilary Mullen	Wednesday	G5.04
25000080	Joe Bloggs	Amersham	BSc SFT Software Technologies	CO450	Computer Architectures	Justin Luker	Monday	G5.02
25000080	Joe Bloggs	Amersham	BSc SFT Software Technologies	CO452	Programming Concepts	Richard Jones	Tuesday	G5.03
25000080	Joe Bloggs	Amersham	BSc SFT Software Technologies	CO455	User Experience	Kevin Maher	Tuesday	G5.08

Normalization – Online Order Placing

Customer Order Bill				
Customer ID	1001		Order Id	1001
Customer Name	SunBeam Infotech		Order Date	16-Apr-2018
Customer Address	Plot R/2		Delivery Date	20-Apr-2018
	Marketyard Road			
	Pune			
	411037			
Phone	020-24260308			
Product Id	**Product Name**	**Quantity**	**Product Rate**	**Product Total**
101	Notebook	4	20	80
102	Pencil	10	2	20
103	File	2	25	50
			Bill Total	**150**

Paper Work		
1.	CID	Int, compulsory
2.	CNAME	alphabets, compulsory
3.	CADDR	e.g. pin – 6 digits, colmulsory
4.	CPHONE	16 digits
5.	OID	Int
6.	ODATE	Date and time
7.	ODELDATE	Date
8.	PID	Int
9.	PNAME	Chars
10.	PRATE	Fractional values
11.	PQTY	Int (cannot be –ve)
12.	PTOTAL	…
13.	FTOTAL	…

CUSTOMER_PLACE_ORDER		
1.	CID	INT
2.	CNAME	VARCHAR(40)
3.	CAREA	VARCHAR(20)
4.	CSTREET	VARCHAR(20)
5.	CCITY	VARCHAR(20)
6.	CPIN	CHAR(6)
7.	CPHONE	VARCHAR(14)
8.	OID	INT
9.	ODATE	DATETIME
10.	ODELDATE	DATE
11.	PID	INT
12.	PNAME	VARCHAR(40)
13.	PRATE	DECIMAL(7,2)
14.	PQTY	INT
15.	PTOTAL	DECIMAL(7,2)
16.	FTOTAL	DECIMAL(7,2)

CUSTOMER_PLACE_ORDER --> UNF		
1.	CID	INT
2.	CNAME	VARCHAR(40)
3.	CAREA	VARCHAR(20)
4.	CSTREET	VARCHAR(20)
5.	CCITY	VARCHAR(20)
6.	CPIN	CHAR(6)
7.	CPHONE	VARCHAR(14)
8.	OID	INT --> PK
9.	ODATE	DATETIME
10.	ODELDATE	DATE
11.	PID	INT
12.	PNAME	VARCHAR(40)
13.	PRATE	DECIMAL(7,2)
14.	PQTY	INT

Order Id	Product Id	Product Name	Quantity	Product Rate	Product Total
1001	101	Notebook	4	20	80
1001	102	Pencil	10	2	20
1001	103	File	2	25	50

ORDERED_PRODUCTS		
1.	PID	INT --> PK
2.	PNAME	VARCHAR(40)
3.	PRATE	DECIMAL(7,2)
4.	PQTY	INT

ORDERED_PRODUCTS <-- 1-NF		
1.	OID	INT --> CPK
2.	PID	INT --> CPK
3.	PNAME	VARCHAR(40)
4.	PRATE	DECIMAL(7,2)
5.	PQTY	INT

CUSTOMER_ORDERS <-- 1-NF		
1.	CID	INT
2.	CNAME	VARCHAR(40)
3.	CAREA	VARCHAR(20)
4.	CSTREET	VARCHAR(20)
5.	CCITY	VARCHAR(20)
6.	CPIN	CHAR(6)
7.	CPHONE	VARCHAR(14)
8.	OID	INT --> PK
9.	ODATE	DATETIME
10.	ODELDATE	DATE

ORDER_DETAILS <-- 2-NF		
1.	OID	INT <-- CPK
2.	PID	INT <-- CPK
3.	PQTY	INT

PRODUCTS <-- 2-NF		
1.	PID	INT <-- PK
2.	PNAME	VARCHAR(40)
3.	PRATE	DECIMAL(7,2)

CUSTOMER_ORDERS <-- 2-NF		
1.	CID	INT
2.	CNAME	VARCHAR(40)
3.	CAREA	VARCHAR(20)
4.	CSTREET	VARCHAR(20)
5.	CCITY	VARCHAR(20)
6.	CPIN	CHAR(6)
7.	CPHONE	VARCHAR(14)
8.	OID	INT <-- PK
9.	ODATE	DATETIME
10.	ODELDATE	DATE

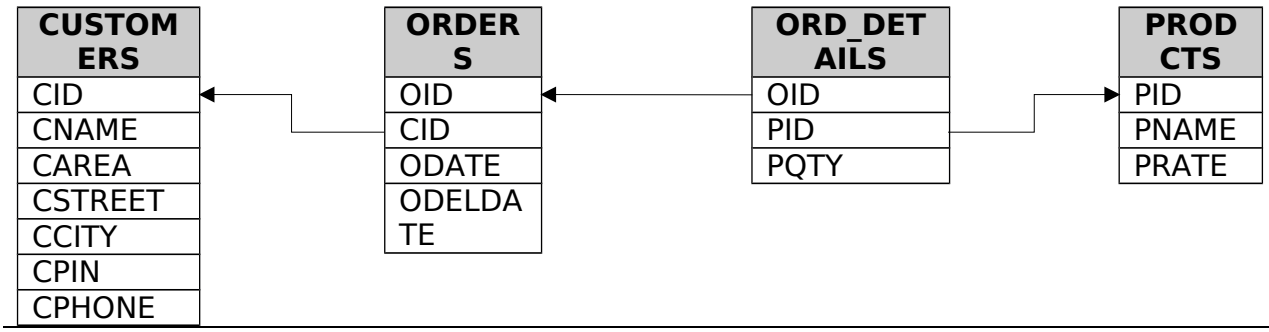
CUSTOMERS			
1.	CID	INT	PK
2.	CNAME	VARCHAR(40)	
3.	CAREA	VARCHAR(20)	
4.	CSTREET	VARCHAR(20)	
5.	CCITY	VARCHAR(20)	
6.	CPIN	CHAR(6)	
7.	CPHONE	VARCHAR(14)	

ORDERS			
1.	CID	INT	FK
2.	OID	INT	PK
3.	ODATE	DATETIME	
4.	ODELDATE	DATE	

ORDER_DETAILS			
1.	OID	INT	FK
2.	PID	INT	FK
3.	PQTY	INT	

PRODUCTS			
1.	PID	INT	PK
2.	PNAME	VARCHAR(40)	
3.	PRATE	DECIMAL(7,2)	

ER-DIAGRAM



CUSTOMERS			
1.	CID	INT	PK
2.	CNAME	VARCHAR(40)	
3.	CAREA	VARCHAR(20)	
4.	CSTREET	VARCHAR(20)	
5.	CPIN	CHAR(6)	FK
6.	CPHONE	VARCHAR(14)	

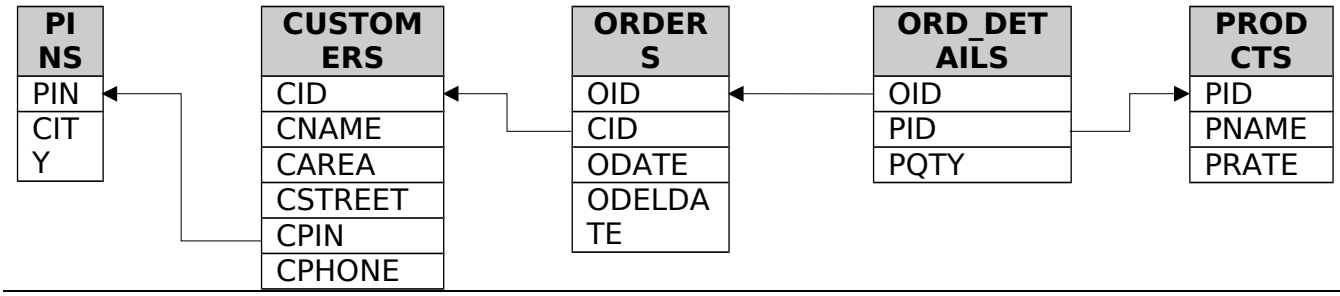
PINS			
1.	PIN	CHAR(6)	PK
2.	CITY	VARCHAR(20)	

ORDERS			
1.	CID	INT	FK
2.	OID	INT	PK
3.	ODATE	DATETIME	
4.	ODELDATE	DATE	

ORDER_DETAILS			
1.	OID	INT	CPK | FK
2.	PID	INT	CPK | FK
3.	PQTY	INT	

PRODUCTS			
1.	PID	INT	PK
2.	PNAME	VARCHAR(40)	
3.	PRATE	DECIMAL(7,2)	

ER-DIAGRAM



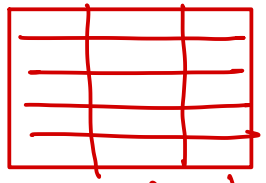
all RDBMS → SQL

↓

Relational

↓

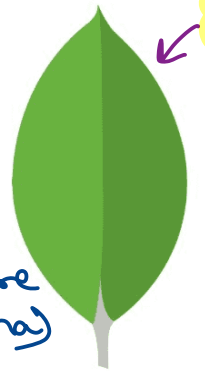
tabular [T1 ↔ T2]



Rows & cols

↑

- fixed structure (schema)
- 100s of GB
- high consistency / transactions.



mongo

↑

cassandra

neo 4j

↑

redis

- flexible schema
- 100s of TB / PB (scaling)
  - data
  - clients
- economical
- eventual consistency

NoSQL Databases

→ no standard query language X

Trainer: Mr. Nilesh Ghule

Not Only SQL

Document oriented databases

Java Script Object Notation

- 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
  "author" : "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

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) → faster access.
- 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.

PBs 24x7

JSON

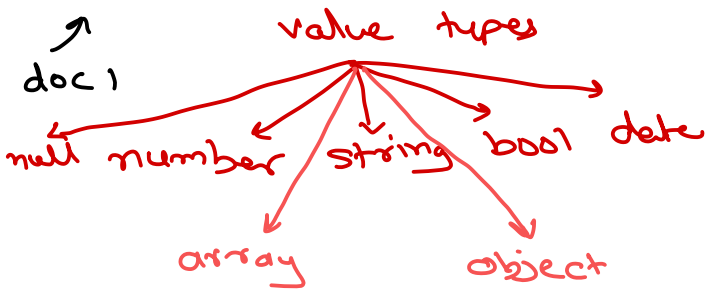
- Java Script Object Notation
- Hierarchical way of organizing data
- Mongo stores JSON data into Binary form. → data type internally identified by a number.

people → collection

doc 1

```
{
    "name" : "Nitin",
    "mobile" : "9881208115",
    "pin" : 411037,
    "sal" : 50000.00
}
```

key, value

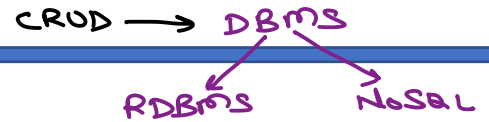


doc 2

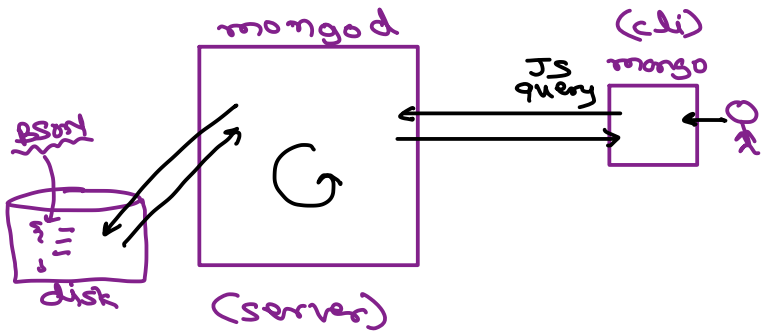
```
{
    "name" : "Nilesh",
    "addr" : {
        "area" : "katraj",
        "city" : "pune",
        "pin" : 411046
    },
    "hobbies" : ["programming", "reading", "cooking"],
    "age" : 37,
    "sal" : 30000.0,
    "political" : false,
    "rating" : null
}
```

string, json object, array, number, boolean, null

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)]

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

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

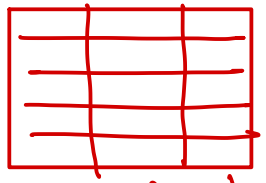
all RDBMS → SQL

↓

Relational

↓

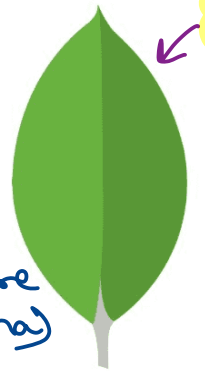
tabular [T1 ↔ T2]



Rows & cols

↑

- fixed structure (schema)
- 100s of GB
- high consistency / transactions.



mongo

↑

cassandra

neo 4j

↑

redis

- flexible schema
- 100s of TB / PB (scaling)
  - data
  - clients
- economical
- eventual consistency

NoSQL Databases

→ no standard query language X

Trainer: Mr. Nilesh Ghule

Not Only SQL

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
  - 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
  - NoSQL
  - NewSQL

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 → each nosql db has its own lang.
- No predefined schema, Unstructured and unpredictable data → images, audio, video, ...
- Eventual consistency rather ACID property
- Based on CAP Theorem (Brewer's).
- Prioritizes high performance, high availability and scalability
- BASE Transaction
  - Basically Available → basic service is run 24x7.
  - Soft state → data is auto adjusted as per cluster size. flexible schema.
  - Eventual consistency → changes will be visible all clients eventually (not immediately).

1970 - RDBMS concepts
⋮
1980 - RDBMS - oracle
⋮

1983 - internet
- static pages.

1995 - Java applets
- dynamic pages

internet - business
common
"

data burst

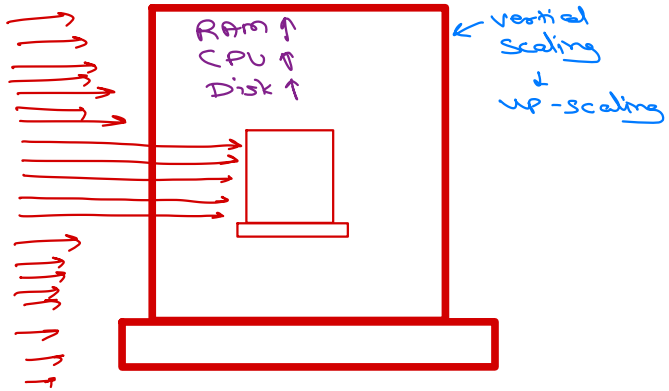
1998 - Carl
↳ No SQL
↳ made his own database.

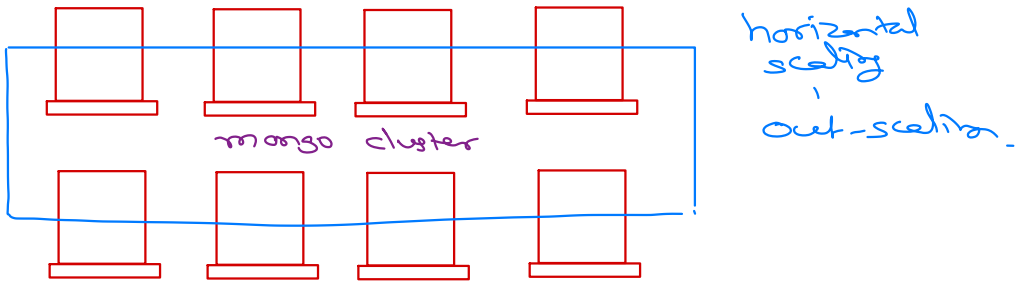
high volume of data
variety data (flexi)
scalability

2004 : global conference
- twitter : #nosql

Scaling

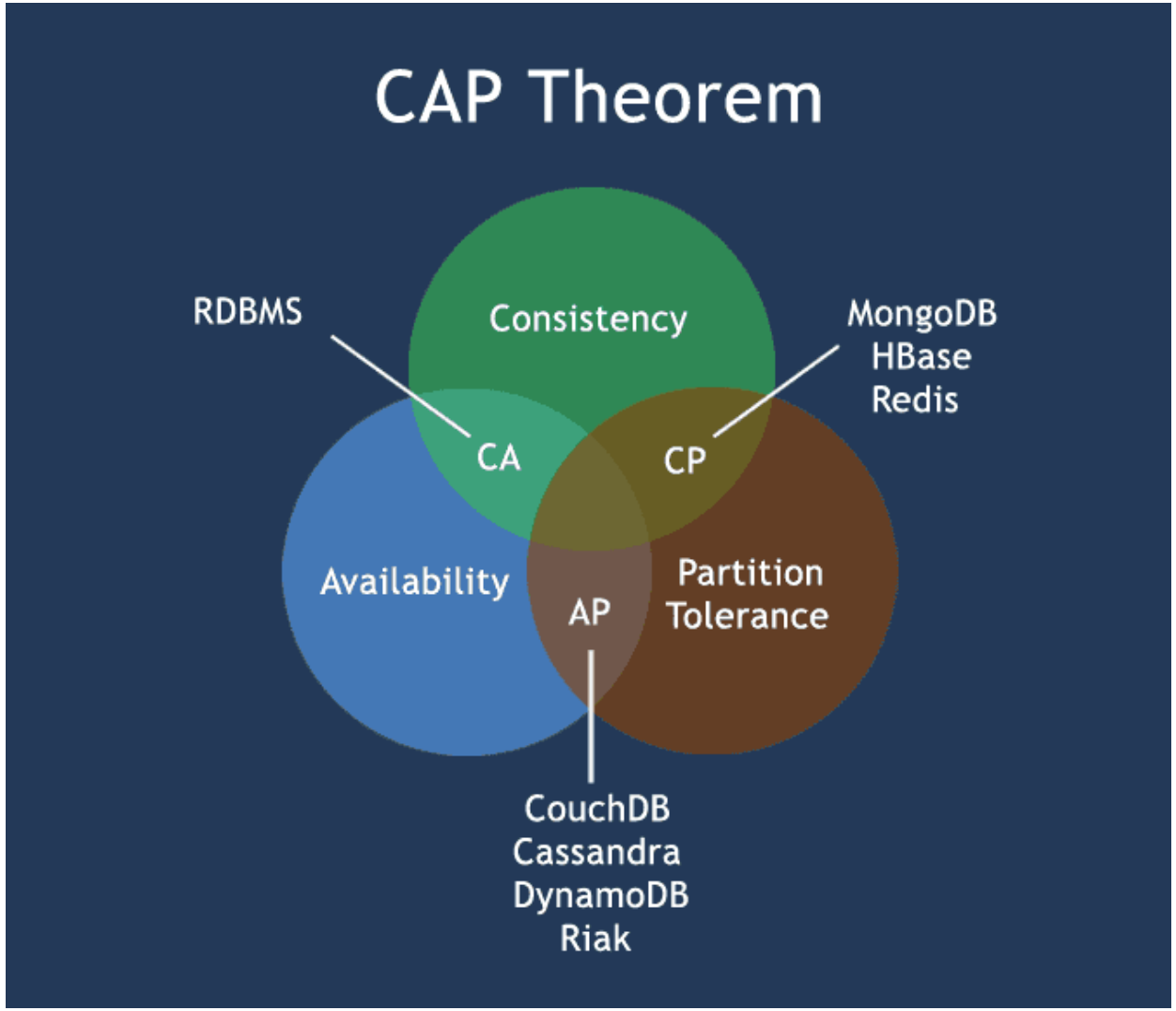
- Scalability is the ability of a system to expand to meet your business needs.
- E.g. scaling a web app is to allow more people to use your application.
- Types of scaling
  - Vertical scaling: Add resources within the same logical unit to increase capacity. E.g. add CPUs to an existing server, increase 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.

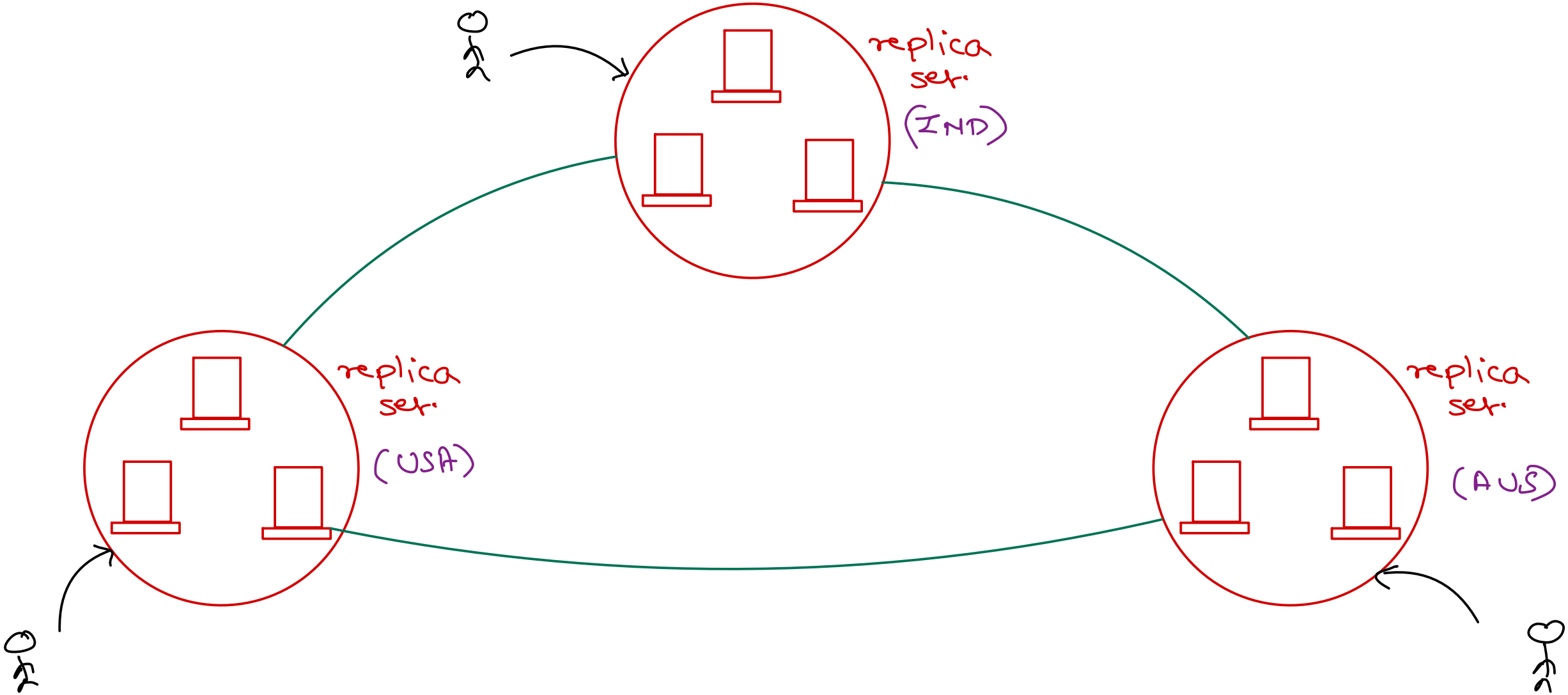




CAP (Brewer's) Theorem

- **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.




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

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.

Applications

- When to use NoSQL?
  - Large amount of data (TBs)
  - Many Read/Write ops
  - Economical Scaling → horizontal 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

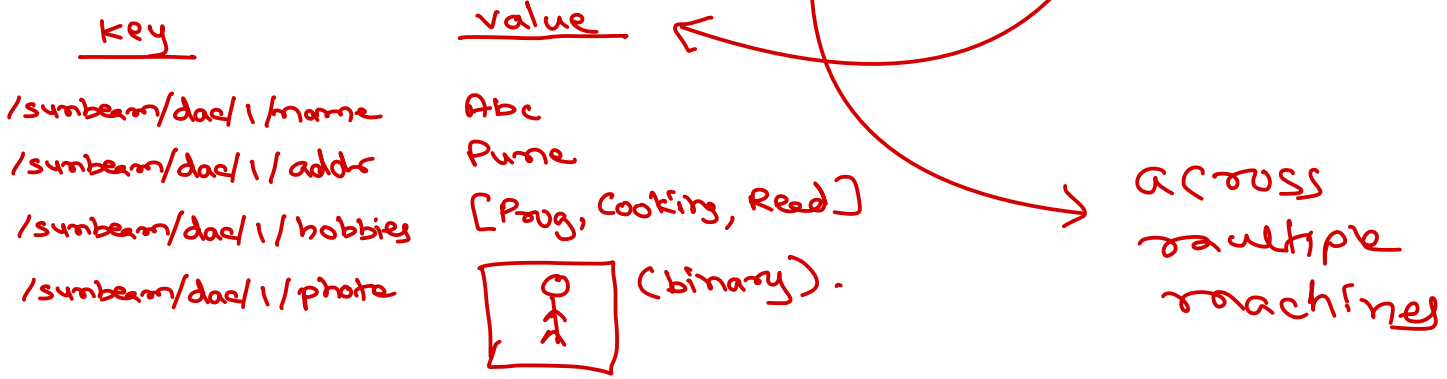
	RDBMS	NoSQL
Types	All types support SQL standard	Multiple types exists, such as document stores, key value stores, column databases, etc
History	Developed in 1970	Developed in 2000s
Examples	SQL Server, Oracle, MySQL	MongoDB, HBase, Cassandra, Redis, Neo4J
Data Storage Model	Data is stored in rows and 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	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. → non-tabular
- There is no standardization/rules of how NoSQL database to be designed.
- All available NoSQL databases can be broadly categorized as follows:
  1. Key-value databases
  2. Column-oriented databases
  3. Graph databases
  4. Document oriented databases → mongo

Key-value database

- Based on Amazon's Dynamo database.
- For handling huge data of any type. → GB, TB
- 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, ...

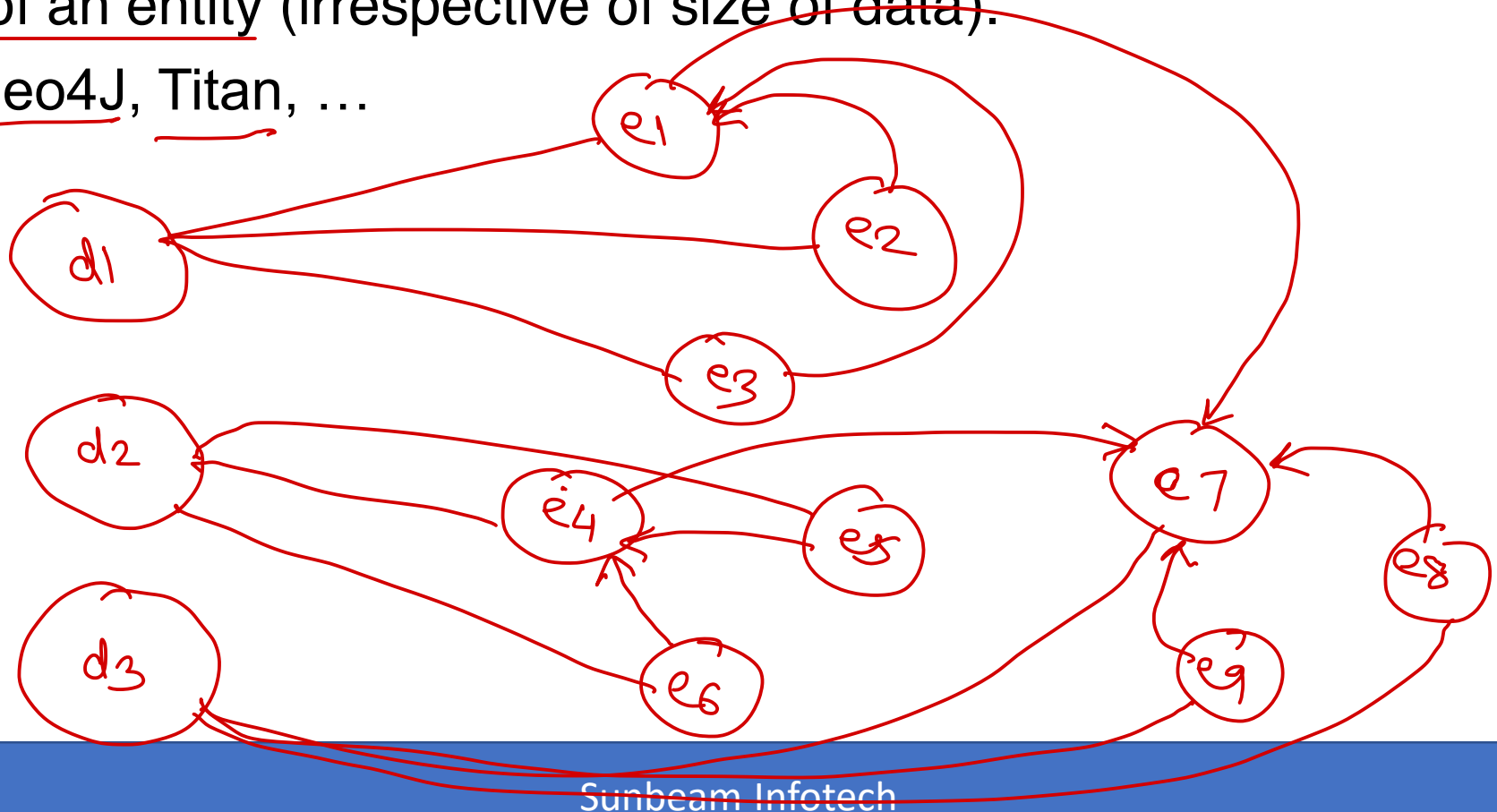


Column-oriented databases

- 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

Java Script Object Notation

- 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
  "author" : "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

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>