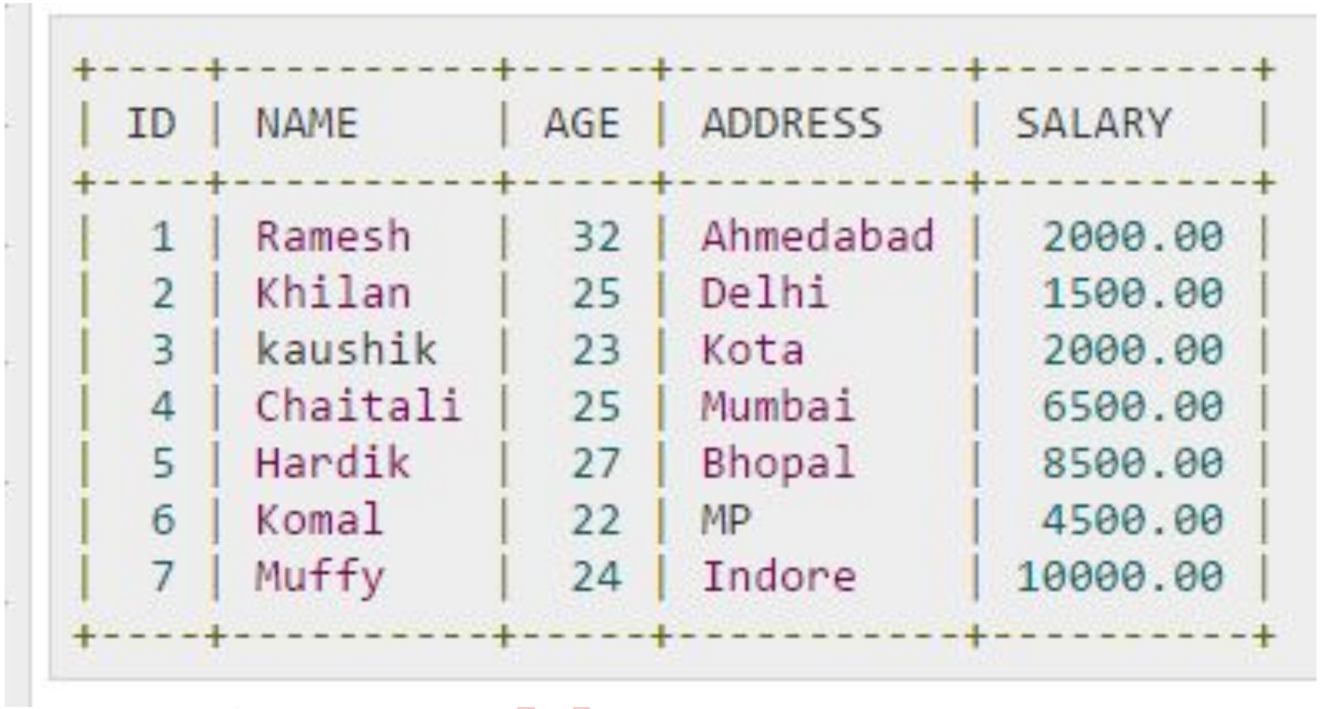
Database Management System
MSMBAinUSA.com

## DBMS Terminology

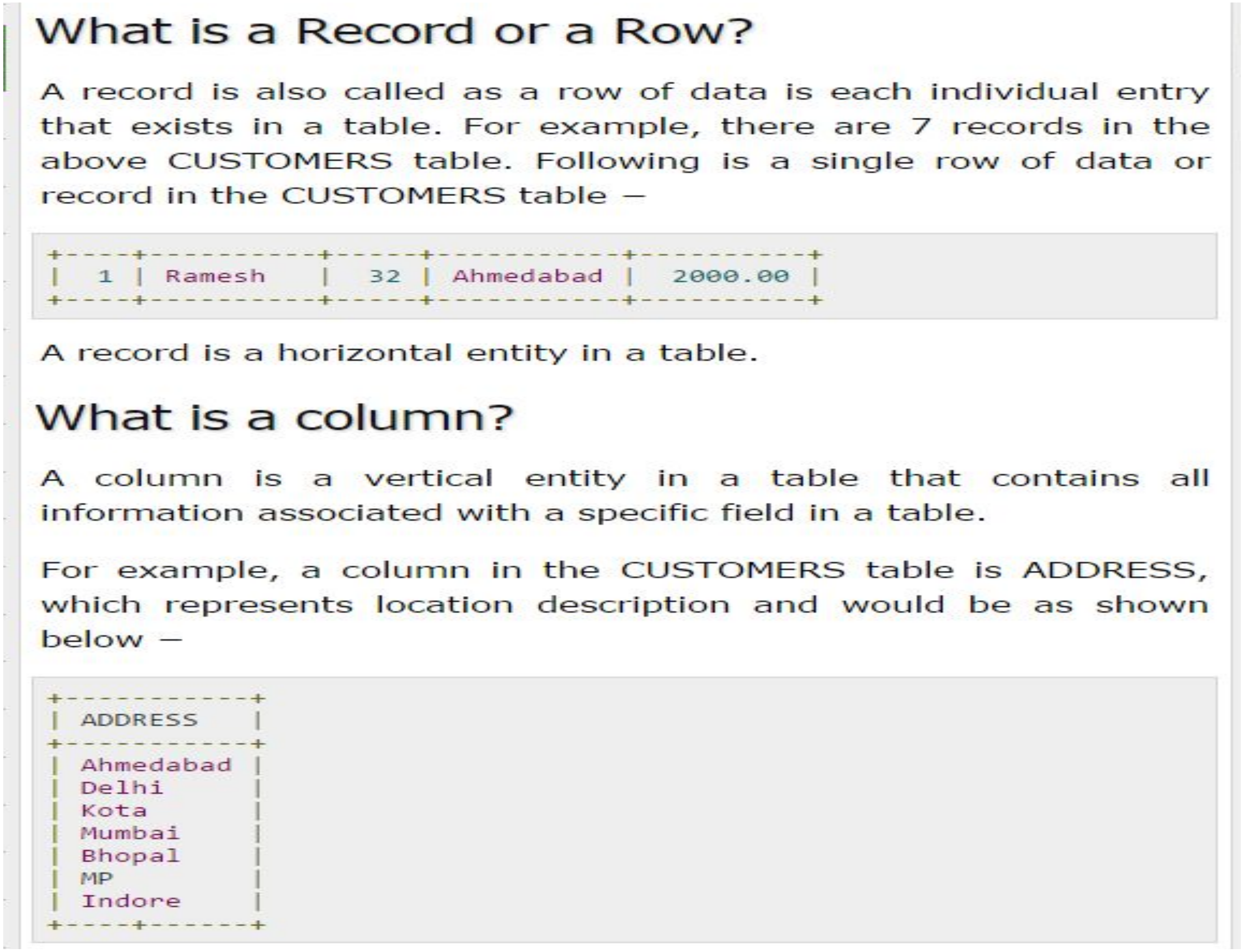
- What is a table?
- The data in an RDBMS is stored in database objects which are called as **tables**.
- This table is basically a collection of related data entries and it consists of numerous columns and rows.
- Remember, a table is the most common and simplest form of data storage in a relational database.
- The following program is an example of a CUSTOMERS table –



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

## What is a field?

- Every table is broken up into smaller entities called fields.
- The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.
- A field is a column in a table that is designed to maintain specific information about every record in the table.



## What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table –

```
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

## What is a column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below –

```
+-----------+
| ADDRESS   |
+-----------+
| Ahmedabad |
| Delhi     |
| Kota      |
| Mumbai    |
| Bhopal    |
| MP        |
| Indore    |
+----+------+
```

## What is a NULL value?

- A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.
- It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.
- A field with a NULL value is the one that has been left blank during a record creation.

## SQL Constraints

- Constraints are the rules enforced on data columns on a table.
- These are used to limit the type of data that can go into a table.
- This ensures the accuracy and reliability of the data in the database.
- Constraints can either be column level or table level.
- Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.
- Following are some of the most commonly used constraints available in SQL –

- NOT NULL Constraint − Ensures that a column cannot have a NULL value.
- DEFAULT Constraint − Provides a default value for a column when none is specified.
- UNIQUE Constraint − Ensures that all the values in a column are different.
- PRIMARY Key − Uniquely identifies each row/record in a database table.
- FOREIGN Key − Uniquely identifies a row/record in any another database table.
- CHECK Constraint − The CHECK constraint ensures that all values in a column satisfy certain conditions.

## Entity-Set and Keys

- Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.
- For example, the roll_number of a student makes him/her identifiable among students.
- Following are the different type of keys are available:

1. Primary key
2. Foreign Key
3. Candidate key
4. Super key

## Relational Data Manipulation

- A **data manipulation language** (**DML**) is a family of syntax elements similar to a computer programming language used for selecting, inserting, deleting and updating data in a database.
- Performing read-only queries of data is sometimes also considered a component of DML.
- A popular data manipulation language is that of Structured Query Language (SQL), which is used to retrieve and manipulate data in a relational database.

- Data manipulation languages have their functional capability organized by the initial word in a statement, which is almost always a verb.

- In the case of SQL, these verbs are:

- *SELECT ... FROM ... WHERE ...*
- *INSERT INTO ... VALUES ...*
- *UPDATE ... SET ... WHERE ...*
- *DELETE FROM ... WHERE ...*

## DML Statements

- Viewing the structure of a table insert, update, delete,
- Select all columns,
- specific columns,
- unique records,
- conditional select,
- in clause, between clause, limit,
- aggregate functions (count, min, max, avg, sum),
- group by clause,
- having clause.

## DML Statements

- DML stands for **Data Manipulation Language.**
- DML statements are used to work with the data in tables.
- When you are connected to most multi-user databases, you are in effect working with a private Copy of your tables that can't be seen by anyone else until you are finished.
- The SELECT Statement is considered to be part of DML even though it just retrieves data Rather than modifying it.
- All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

- **Viewing the structure of a table**

- **SQL SELECT Statement:**
- SELECT column1, column2....columnN FROM table_name;

- **SQL WHERE Clause:**
- SELECT column1, column2....columnN FROM table_name WHERE CONDITION;

- **SQL IN Clause:**
- SELECT column1, column2....columnN FROM table_name WHERE column_name IN (val-1, val-2,...val-N);

- **SQL INSERT INTO Statement:**
- INSERT INTO table_name( column1, column2....columnN) VALUES ( value1, value2....valueN);

- **SQL UPDATE Statement:**
- UPDATE table_name SET column1 = value1, column2 = value2....columnN=valueN [ WHERE CONDITION ];

- **SQL DELETE Statement:**
- DELETE FROM table_name WHERE [CONDITION];

- **SQL BETWEEN Clause:**
- SELECT column1, column2....columnN FROM table_name WHERE column_name BETWEEN val-1 AND val-2;

- **SQL LIKE Clause:**
- SELECT column1, column2....columnN FROM table_name WHERE column_name LIKE { PATTERN };

- **SQL ORDER BY Clause:**
- SELECT column1, column2....columnN FROM table_name WHERE CONDITION ORDER BY column_name {ASC|DESC};

- **SQL COUNT Clause:**
- SELECT COUNT(column_name) FROM table_name WHERE CONDITION;

## TCL Statements

- TCL is stands for **Transaction Control Statements.**
- It deals with transaction within a database.
- COMMIT- commits a transaction
- ROLLBACK- Rollback a transaction in case of any errors
- SAVEPOINT- to rollback the transaction making points within groups.

- **SQL ROLLBACK Statement:**
- ROLLBACK;

- **SQL SAVEPOINT Statement:**
- Savepoint sp1;

- **SQL COMMIT Statement:**
- COMMIT;

## DCL statements

- DCL stands for **Data Control Language.**
- It includes commands such as GRANT, REVOKE, Mostly concerned with rights , permissions of the database system.
- **GRANT-** allow users access privileges to database.
- **REVOKE-** withdraw users access privileges to database given by using **GRANT** command.

## Difference between Delete, Truncate and Drop command in SQL

- **Delete:** if we use delete command in a query than it simply deletes all data in a table not structure of the table.

  SQL> delete table_name;

  and all data can recover using rollback command.

  SQL> rollback;
- **Drop:** if we use drop command in a query than it deletes all data and structure of the table.

  SQL> drop table table_name;

- **Truncate:** if we use truncate command in a query than it deletes all data permanently and not structure of the table.

  SQL> truncate table table_name;

  and data can not recover using rollback command.

  Another difference between delete and truncate command:
- if we are using auto increment variable in a table and we run delete command, after delete command if we insert any row than it simply increments the value of auto increment variable but in truncate command it starts value from one.

# DDL Statements

- Creating Databases,
- Using Databases,
- datatypes,
- Creating Tables (with integrity constraints – primary key, default, check, not null),
- Altering Tables,
- Renaming Tables,
- Dropping Tables,
- Truncating Tables,
- Backing Up and Restoring databases

## Types of Statements

- There are basic two major categories of SQL statement:- DDL(Data Definition language) and DML(Data Manipulation language).
- **DDL Statements**
- **DML Statements**
- **TCL Statements**
- **DCL Statements**

## DDL Statements

- DDL stands for **Data Definition Language.**
- DDL statements are used to build and modify the structure of tables and other objects in the database.
- When you execute a DDL statement, it takes effect immediately.
- SQL uses the following set of commands to define database schema –

## 1. Creating Databases

- Create database Patkar_college;
- Create database FYCS;

- Show database;

Patkar_college,
FYCS

- Using database:
- Use FYCS

## Datatypes

- A **data type** is a set of **data** with values having predefined characteristics.
- Examples of **data types** are: integer, floating point unit number, character, string, and pointer.
- Properly defining the fields in a table is important to the overall optimization of your database.
- You should use only the type and size of field you really need to use;
- don't define a field as 10 characters wide if you know you're only going to use 2 characters.
- These types of fields (or columns) are also referred to as data types, after the **type of data** you will be storing in those fields.

- MySQL uses many different data types broken into three categories: numeric, date and time, and string types.
- **Numeric Data Types:** MySQL uses all the standard ANSI SQL numeric data types.
- The following list shows the common numeric data types and their descriptions:

1. **INT**
2. **FLOAT**
3. **DOUBLE**
4. **DECIMAL**

- **Date and Time Types:**

1. **DATE -** A date in YYYY-MM-DD format.
2. **TIME** - Stores the time in HH:MM:SS format.

- **String Types:** most data you'll store will be in string format. This list describes the common string datatypes in MySQL.

1. **CHAR**
2. **VARCHAR**

## The SQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a table in a database.
- Tables are organized into rows and columns; and each table must have a name.
- SQL CREATE TABLE Syntax:
- CREATE TABLE *table_name*
  (
  *column_name1 data_type*(*size*),
  *column_name2 data_type*(*size*),
  *column_name3 data_type*(*size*),
  ....
  );

- The column_name parameters specify the names of the columns of the table.
- The data_type parameter specifies what type of data the column can hold (e.g. varchar, integer, decimal, date, etc.).
- The size parameter specifies the maximum length of the column of the table.

## SQL CREATE TABLE + CONSTRAINT Syntax

-
  ```
  CREATE TABLE table_name
  (
  column_name1 data_type(size) constraint_name,
  column_name2 data_type(size) constraint_name,
  column_name3 data_type(size) constraint_name,
  ....
  );
  ```
- In SQL, we have the following constraints:
- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **CHECK** - Ensures that the value in a column meets a specific condition
- **DEFAULT** - Specifies a default value for a column.

## Altering Tables

- The SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table.
- You would also use ALTER TABLE command to add and drop various constraints on a an existing table.
- **Syntax:**
- The basic syntax of **ALTER TABLE** to **add** a new column in an existing table is as follows:
- **ALTER TABLE table_name ADD column_name datatype;**
- The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows:
- ALTER TABLE table_name DROP COLUMN column_name;

- The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows:
- ALTER TABLE table_name MODIFY COLUMN column_name datatype;

- The basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows:
- ALTER TABLE table_name MODIFY column_name datatype NOT NULL;

- The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows:
- ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);

- The basic syntax of ALTER TABLE to **ADD CHECK CONSTRAINT**to a table is as follows:
- ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);

- The basic syntax of ALTER TABLE to **ADD PRIMARY KEY**constraint to a table is as follows:
- ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1);

- The basic syntax of ALTER TABLE to **DROP CONSTRAINT** from a table is as follows:
- ALTER TABLE table_name DROP CONSTRAINT MyUniqueConstraint;

- If you're using MySQL, the code is as follows:
- ALTER TABLE table_name DROP INDEX MyUniqueConstraint;

- The basic syntax of ALTER TABLE to **DROP PRIMARY KEY** constraint from a table is as follows:
- ALTER TABLE table_name DROP CONSTRAINT MyPrimaryKey;

- If you're using MySQL, the code is as follows:
- ALTER TABLE table_name DROP PRIMARY KEY;

## Renaming a Table

- To rename a table, use the **RENAME** option of the ALTER TABLE statement.
- To rename ABC_tbl to PQR_tbl.
- mysql> ALTER TABLE ABC_tbl RENAME TO PQR_tbl;

## SQL DROP TABLE Statement:

- It is very easy to drop an existing MySQL table, but you need to be very careful while deleting any existing table because data lost will not be recovered after deleting a table.
- DROP TABLE table_name;

## SQL - TRUNCATE TABLE Command

- The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.
- You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.
- **Syntax:**
- The basic syntax of **TRUNCATE TABLE** is as follows:
- TRUNCATE TABLE table_name;

## Following is the example to truncate:

- TRUNCATE TABLE CUSTOMERS;
- Now, CUSTOMERS table is truncated and following would be the output from SELECT statement:

```
SQL> SELECT * FROM CUSTOMERS;
Empty set (0.00 sec)
```

## Creating A Backup

- The ***mysqldump*** command is used to create textfile "dumps" of databases managed by MySQL. These dumps are just files with all the SQL commands needed to recreate the database from scratch. The process is quick and easy.
- If you want to back up a **single database**, you merely create the dump and send the output into a file, like so:
- mysqldump database_name > database_name.sql

- **Multiple databases** can be backed up at the same time:
- mysqldump --databases database_one database_two > two_databases.sql

- It is also simple to back up all of the databases on a server:
- mysqldump --all-databases > all_databases.sql

## Restoring a Backup

- Since the dump files are just SQL commands, you can restore the database backup by telling mysql to run the commands in it and put the data into the proper database.
- mysql database_name < database_name.sql
- In the code above, **database_name** is the name of the database you want to restore, and**database_name.sql** is the name of the backup file to be restored..
- If you are trying to restore a single database from dump of all the databases, you have to let mysql know like this:
- mysql --one-database database_name < all_databases.sql

## Transaction Processing

- The Concept of Transaction
- States of Transaction
- Concurrent Execution of Multiple transactions
- Serializability

## What is a Transaction?

- A logical unit of work on a database
  - An entire program
  - A portion of a program
  - A single command
- The entire series of steps necessary to accomplish a logical unit of work.
- Successful transactions change the database from one CONSISTENT STATE to another .

  (One where all data integrity constraints are satisfied)

## Example of a Transaction

- A **transaction** is a discrete unit of work that must be completely processed or not processed at all.
- Example: Transferring funds from a checking account to a saving account.

- Updating a Record
  - Locate the Record on Disk
  - Bring record into Buffer
  - Update Data in the Buffer
  - Writing Data Back to Disk

## DML and DDL Transactions

- The following are the DML and DDL operations supported in a distributed transaction:
- CREATE TABLE
- DELETE
- INSERT
- LOCK TABLE
- SELECT
- SELECT FOR UPDATE
- You can execute DML and DDL statements in parallel, and INSERT direct load statements serially, but note the following restrictions:

- All remote operations must be SELECT statements.
- These statements must not be clauses in another distributed transaction.
- If the table referenced in the *table_expression_clause* of an INSERT, UPDATE, or DELETE statement is remote, then execution is serial rather than parallel.
- You cannot perform remote operations after issuing parallel DML/DDL or direct load INSERT.

- If the transaction begins, it executes serially.
- No loopback operations can be performed on the transaction originating the parallel operation.
- For example, you cannot reference a remote object that is actually a synonym for a local object.
- If you perform a distributed operation other than a SELECT in the transaction, no DML is parallelized.

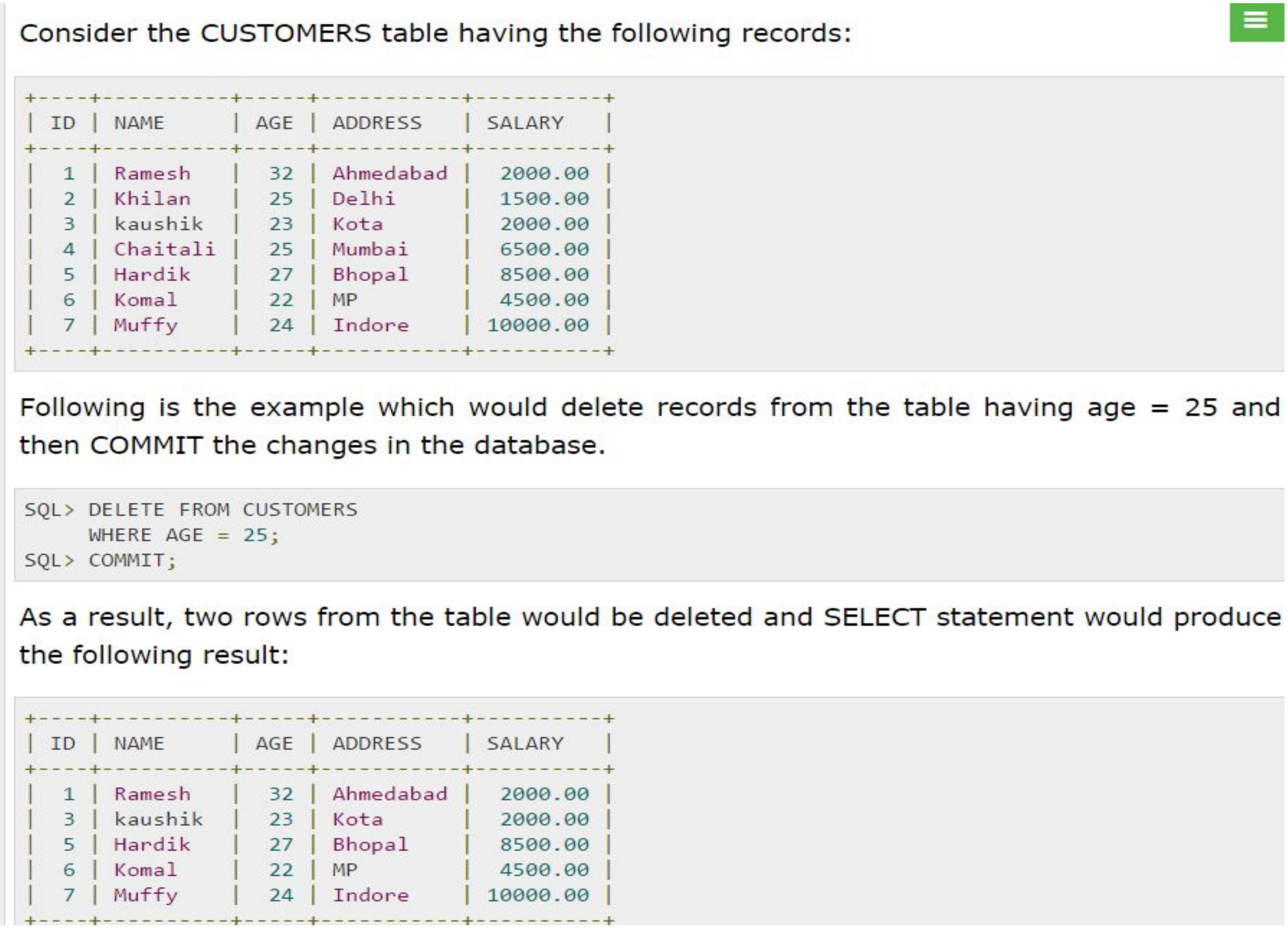
## Transaction Control Statements

- The following list describes supported transaction control statements:
- **COMMIT-** to save the changes.
- **ROLLBACK-** to rollback the changes.
- **SAVEPOINT-** creates points within groups of transactions in which to ROLLBACK.

- Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only.
- They can not be used while creating tables or dropping them because these operations are automatically committed in the database.

## The COMMIT Command:

- The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.
- The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.
- The syntax for COMMIT command is as follows:
- COMMIT;



Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is the example which would delete records from the table having age = 25 and then COMMIT the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
     WHERE AGE = 25;
SQL> COMMIT;
```

As a result, two rows from the table would be deleted and SELECT statement would produce the following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

## The ROLLBACK Command:

- The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.
- The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.
- The syntax for ROLLBACK command is as follows:
- ROLLBACK;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is the example, which would delete records from the table having age = 25 and then ROLLBACK the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
     WHERE AGE = 25;
SQL> ROLLBACK;
```

As a result, delete operation would not impact the table and SELECT statement would produce the following result:

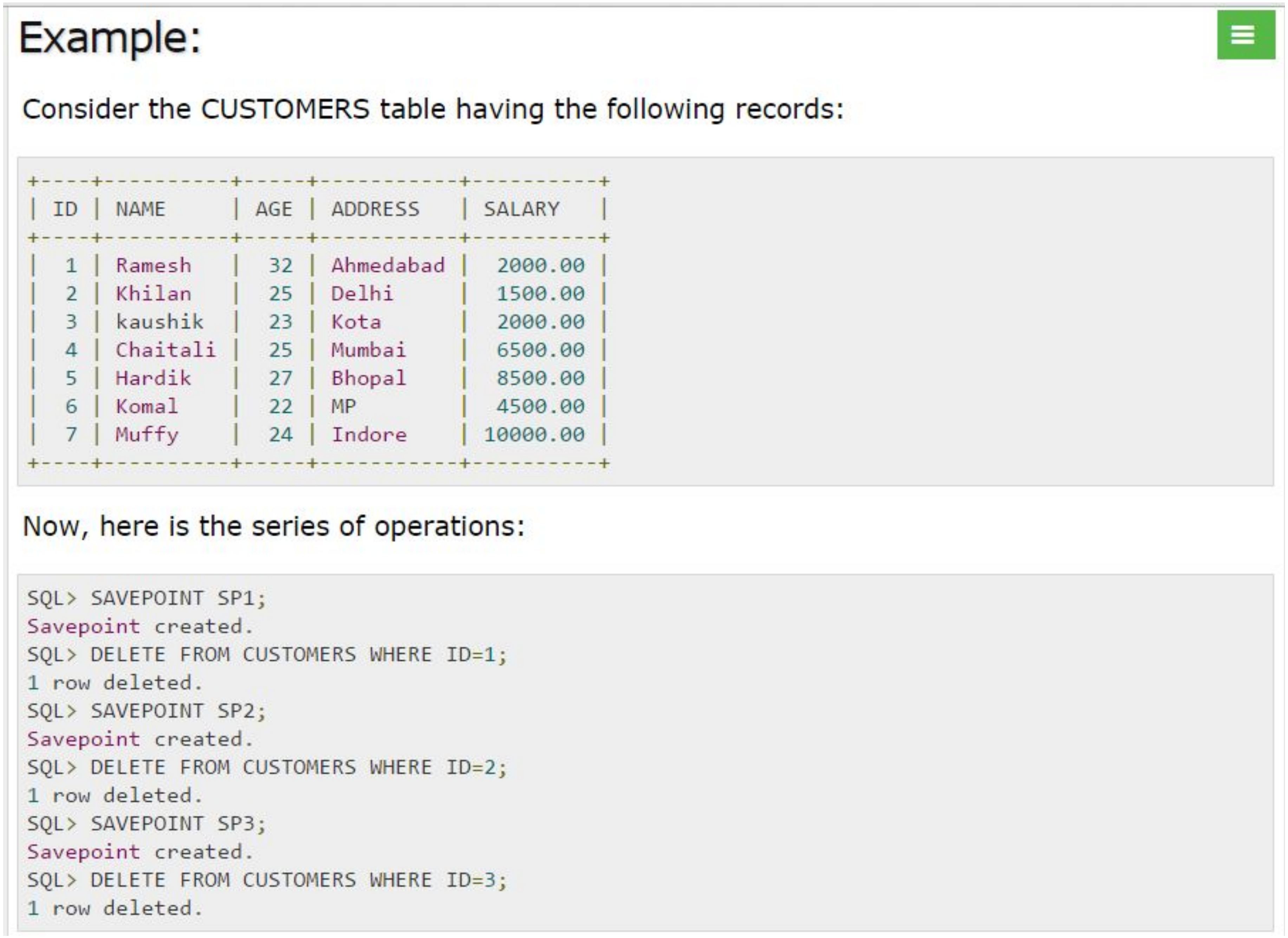
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

## The SAVEPOINT Command:

- A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
- The syntax for **SAVEPOINT** command is as follows:
- SAVEPOINT SAVEPOINT_NAME;

- This command serves only in the creation of a SAVEPOINT among transactional statements.
- The ROLLBACK command is used to undo a group of transactions.

- Following is an example where you plan to delete the three different records from the CUSTOMERS table.
- You want to create a SAVEPOINT before each delete,
- so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state:



## Example:

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

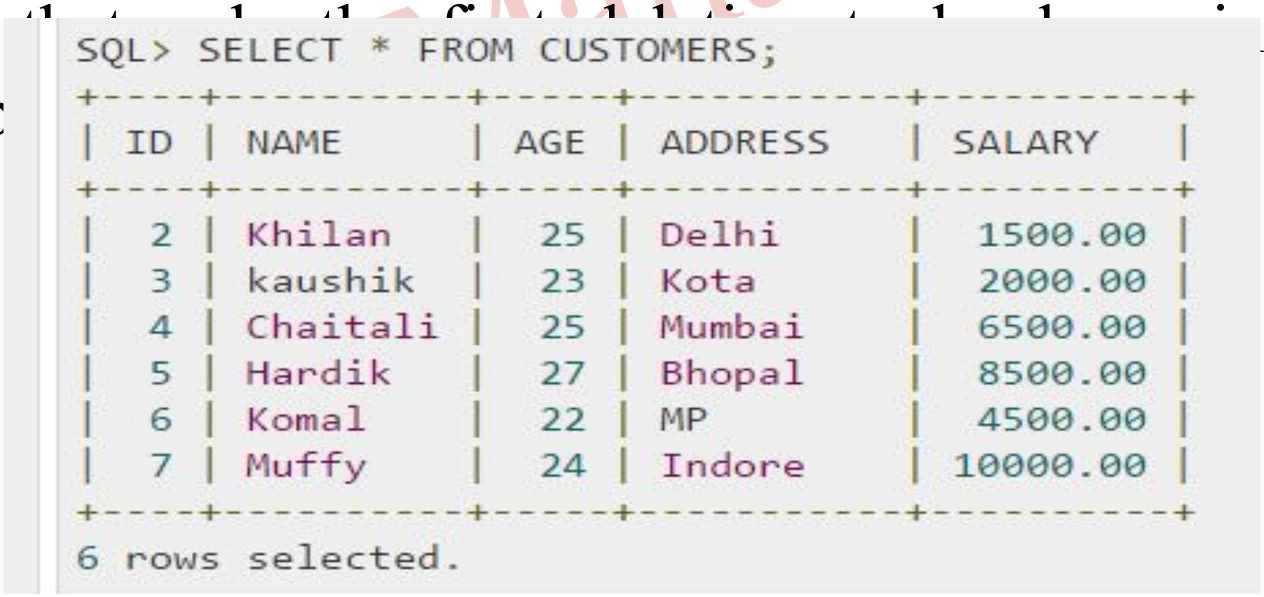
Now, here is the series of operations:

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.
SQL> SAVEPOINT SP2;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted.
SQL> SAVEPOINT SP3;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.
```

- Now that the three deletions have taken place, say you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone:
- SQL> ROLLBACK SP2;

Rollback complete.

- Notice ce you rolled b



```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
6 rows selected.
```

## ACID properties

- To ensure data integrity.
- The database system must maintain the **ACID** properties.
- **The ACID properties:**
- **Atomicity** : A transaction is treated as a single/atomic unit of operation and is either executed completely or not at all
- **Consistency** : A transaction preserves DB consistency, i.e., does not violate any integrity constraints
- **Isolation** : A transaction is executed as if it would be the only one. In database systems, **isolation** determines how **transaction** integrity is visible to other users.
- **Isolation** is typically **defined** at database level as a **property** that defines how/when the changes made by one operation become visible to other.
- **Durability** : The updates of a committed transaction are permanent in the DB.

## Atomicity

- Either all or none of the transaction's operations are performed .
- Partial results of an interrupted transactions must be undone.
- Transaction recovery is the activity of the restoration of atomicity due to input errors, system overloads, and deadlocks.
- Crash recovery is the activity of ensuring atomicity in the presence of system crashes.

## Consistency

- The consistency of a transaction is simply its correctness and ensures that a transaction transforms a consistent DB into a consistent DB.
- Transactions are correct programs and do not violate database integrity constraints.
- Dirty data is data that is updated by a transaction that has not yet committed.

## Isolation

- Isolation is the property of transactions which requires each transaction to see a consistent DB at all times.
- If two concurrent transactions access a data item that is being updated by one of them (i.e., performs a write operation), it is not possible to guarantee that the second will read the correct value.
- Interconsistency of transactions is obviously achieved if transactions are executed serially.
- Therefore, if several transactions are executed concurrently, the result must be the same as if they were executed serially in some order.

- **Example:** Consider the following two transactions, where initially $x = 50$:

```
T1: Read(x)                T2: Read(x)
    x ← x+1                    x ← x+1
    Write(x)                   Write(x)
    Commit                     Commit
```

- Possible execution sequences:

```
T1: Read(x)
T1: x ← x+1
T1: Write(x)
T1: Commit
T2: Read(x)
T2: x ← x+1
T2: Write(x)
T2: Commit
```

  - Serial execution: we get the correct result $x = 52$ (the same for $\{T_2, T_1\}$)

```
T1: Read(x)
T1: x ← x+1
T2: Read(x)
T1: Write(x)
T2: x ← x+1
T2: Write(x)
T1: Commit
T2: Commit
```

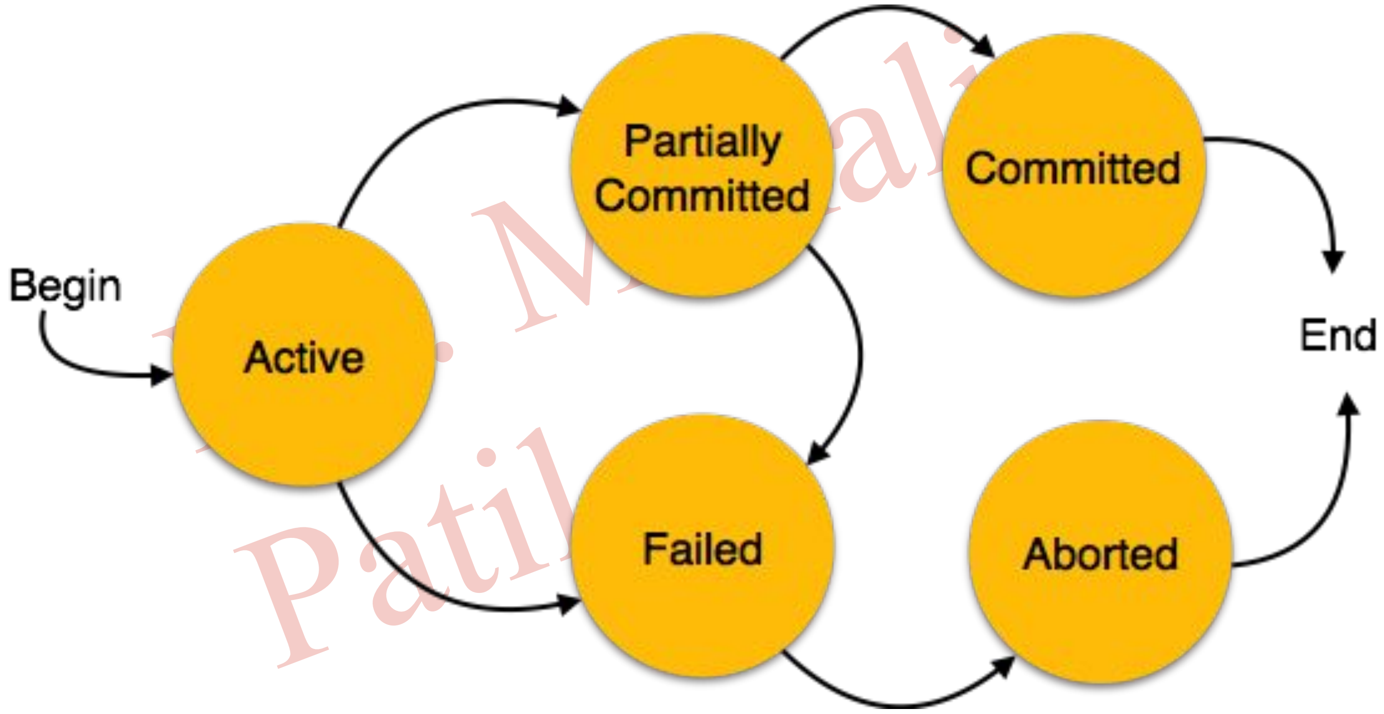
  - Concurrent execution: $T_2$ reads th value of $x$ while it is being changed; the result is $x = 51$ and is incorrect!

## Durability

- Once a transaction commits, the system must guarantee that the results of its operations will never be lost, in spite of subsequent failures.
- Database recovery is used to achieve the task

## States of Transaction

- A transaction in a database can be in one of the following states –



- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

- **Aborted** – If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted.
- The database recovery module can select one of the two operations after a transaction aborts –
  - Re-start the transaction
  - Kill the transaction

- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

## Schedule

- Schedule:

- A schedule is series of transactions.

- A schedule is list of actions (reading, aborting or committing) from a set of transactions.

- The order in which two actions of transaction T appear in a schedule must be the same as the order in which they appear in T.

- A sequence of instructions that specify the chronological order in which instructions of concurrent transactions are executed.

## Example

- The schedule shows an execution order for actions of 2 transactions T1 & T2

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(B) |
| | W(B) |
| R(C) | |
| W(C) | |

Fig. A schedule involving 2 transactions

## Serial schedule

- A schedule is serial if the transactions are executed as a whole one after the other.
- If T2 is scheduled to run after T1 is completely finished then the schedule is serial.
- If T2 is scheduled to run when T! is only halfway finished then the schedule is not serial.
- A serial schedule will preserve the consistency of the database state.
- No interleaving allowed.

## Notation for Transactions and Schedules

- Only the read and writes performed by the transaction matter.
- $r_T(X)$ and $w_T(X)$ will be shorthand for transaction T reads and writes element X.
- The transactions used in examples can be rewritten as:
  $T_1$: $r_1(A)$; $w_1(A)$; $r_1(B)$; $w_1(B)$;
  $T_2$: $r_2(A)$; $w_2(A)$; $r_2(B)$; $w_2(B)$;
- The details of what T1 and T2 are doing (adding versus multiplying) do not matter since we will assume the worse.

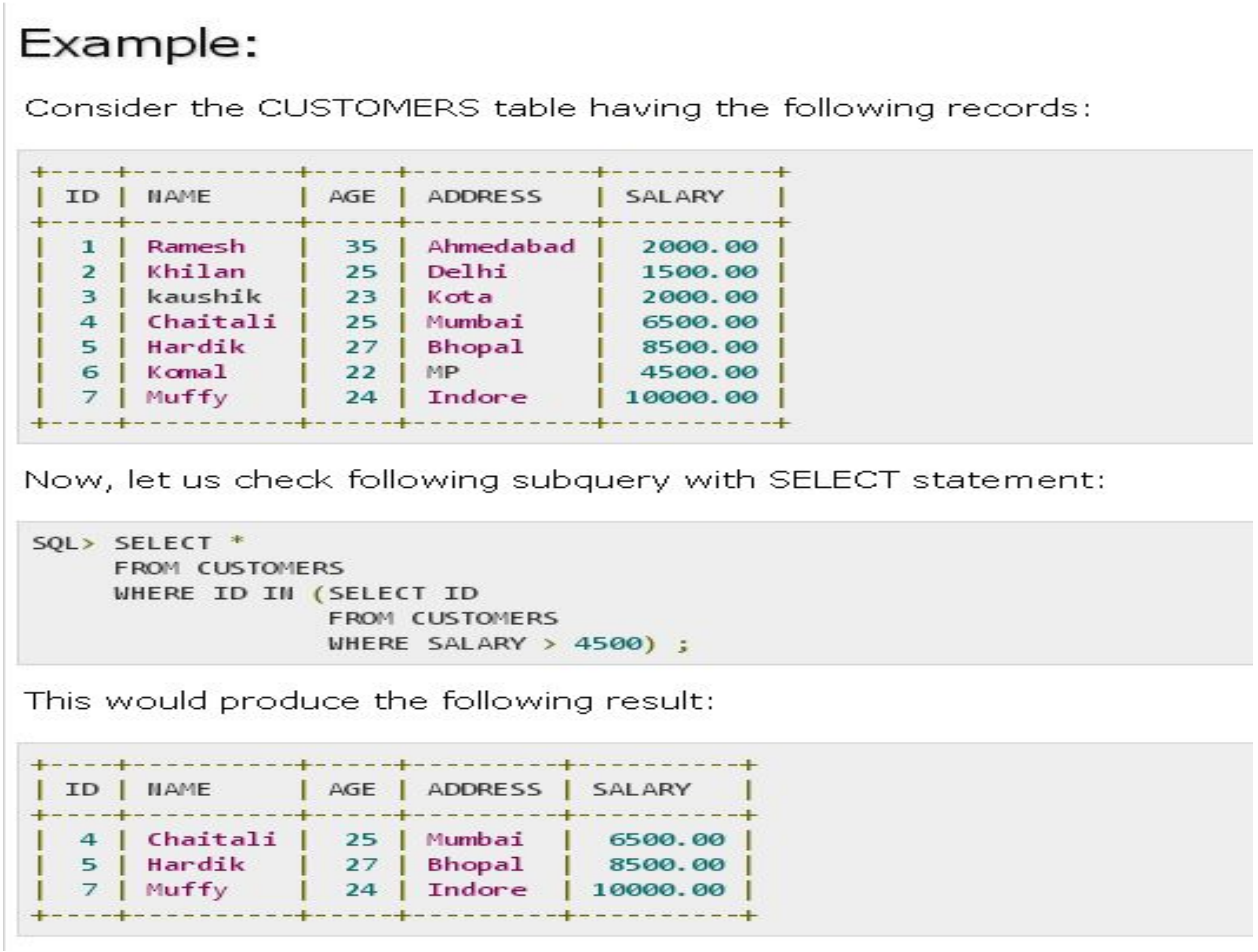
## Subqueries

- A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.
- subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

## Subqueries with the SELECT(IN) Statement:

- Subqueries are most frequently used with the SELECT statement.
- The basic syntax is as follows:

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
(SELECT column_name [, column_name ]
FROM table1 [, table2 ] [WHERE]);
```



## Example:

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now, let us check following subquery with SELECT statement:

```
SQL> SELECT *
     FROM CUSTOMERS
     WHERE ID IN (SELECT ID
                  FROM CUSTOMERS
                  WHERE SALARY > 4500) ;
```

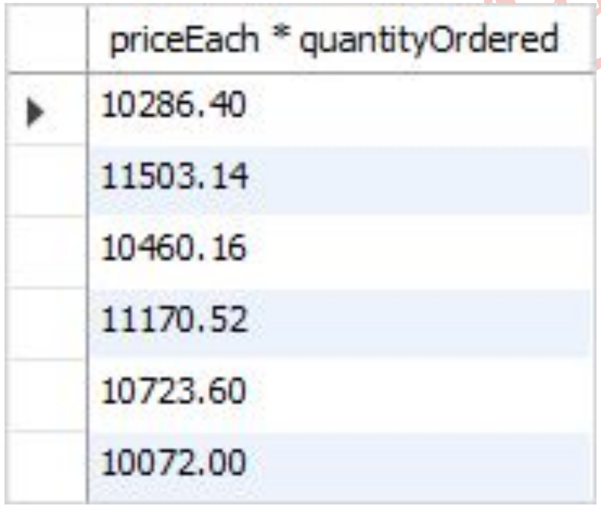
This would produce the following result:

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

## MySQL subquery with EXISTS and NOT EXISTS

- When a subquery is used with EXISTS or NOT EXISTS operator, a subquery returns a Boolean value of TRUE or FALSE.
- The subquery acts as an existence check.
- In the following example, we select a list of customers who have at least one order with total sales greater than 10K.
- First, we build a query that checks if there is, at least, one order with total sales greater than 10K:

- SELECT
priceEach * quantityOrdered
FROM orderdetails
WHERE priceEach * quantityOrdered > 10000



| priceEach * quantityOrdered |
|---|
| 10286.40 |
| 11503.14 |
| 10460.16 |
| 11170.52 |
| 10723.60 |
| 10072.00 |

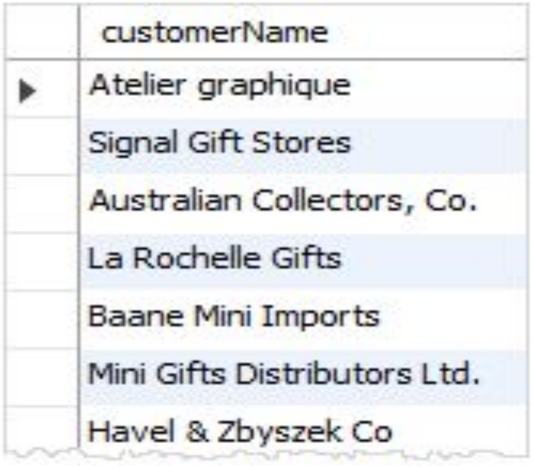
- The query returns 6 rows so that when we use it as a subquery, it will return TRUE;
- therefore the whole query will return all customers:
- SELECT customerName

FROM customers

WHERE EXISTS ( SELECT priceEach * quantityOrdered

FROM orderdetails

WHERE priceEach * quantityOrdered > 10000 )



| customerName |
|---|
| Atelier graphique |
| Signal Gift Stores |
| Australian Collectors, Co. |
| La Rochelle Gifts |
| Baane Mini Imports |
| Mini Gifts Distributors Ltd. |
| Havel & Zbyszek Co |

# ☺ …Thank You… ☺

********