UNIT II
SQL

Database Languages

- Database language is used to handle database.

- Queries of database can be broadly classified as

- Data Definition Language
- Data Manipulation Language
- Data Control Language
- Transaction Control Language

Database Languages

- **Data-Definition Language (DDL)**

➢ The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

➢ Statements are used to define the database structure or schema.

➢ **Example:**

✓ CREATE - to create objects in the database

✓ ALTER - alters the structure of the database

✓ DROP - delete objects from the database

✓ TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed

✓ COMMENT - add comments to the data dictionary

✓ RENAME - rename an object

Database Languages Contd…

- **Data-Manipulation Language (DML)**

➢ The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

➢ Statements are used for managing data within schema objects.

➢ **Example**

- ✓ SELECT - Retrieve data from the database
- ✓ INSERT - Insert data into a table
- ✓ UPDATE - Updates existing data within a table
- ✓ DELETE - deletes all records from a table, the space for the records remain
- ✓ MERGE - UPSERT operation (insert or update)
- ✓ CALL - Call a PL/SQL or Java subprogram
- ✓ EXPLAIN PLAN - explain access path to data
- ✓ LOCK TABLE - control concurrency

Database Languages Contd…

- **Data Control Language (DCL)**

➢ These SQL commands are used for providing security to database objects.

➢ **Example:**

✓ GRANT - gives user's access privileges to database

✓ REVOKE - withdraw access privileges given with the GRANT command

- **Transaction Control Language (TCL)**

➢ Statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

➢ **Example:**

✓ COMMIT - save work done

✓ SAVEPOINT - identify a point in a transaction to which you can later roll back

✓ ROLLBACK - restore database to original since the last

✓ COMMIT SET TRANSACTION - Change transaction options like isolation level and what rollback segment to use

MySQL®

MySQL

- MySQL is a open source, fast, flexible, reliable, RDBMS being used for many small and big businesses.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL is Written in C,C++
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is customizable.
- MySQL supports large databases, up to 50 million rows or more in a table. The default size limit for a table is 4GB, but you can increase it to 8 terabytes (TB).

MySQL Basic Commands

- **To Start MySQL**

```
#mysql –u username –p
Enter password:
```

- **To Access user on Client**

```
#mysql –h Host IP –u username –p
Enter password:
```

- **To Exit MYSQL**

```
    #Exit;     OR    #Quit;
```

- **To check version of MYSQL**

```
    #select version();
```

- **To check current date/time**

```
   #select current_date;
    #select now();
```

MySQL Basic Commands

- **To Create a Database**

  #create database [if not exists] database_name;

- **To Use a Database**

  #use database_name;

- **Displaying Databases**

  #show databases;

- **Removing Databases**

  #drop database [if exists] database_name;

MySQL Data Type

- **Numeric Data Types**

- TINYINT- A very small integer (1 byte)
- SMALLINT- A small integer (2 bytes)
- MEDIUMINT- A medium-sized integer (3 bytes)
- INT- A standard integer (4 bytes)
- BIGINT- A large integer (8 bytes)
- DECIMAL- A fixed-point number (varies)
- FLOAT- A single-precision floating-point number (4 bytes)
- DOUBLE- A double-precision floating-point number (8 bytes)
- BIT- A bit field

MySQL Data Type

- **String Data Types**

➢ CHAR- A fixed-length non-binary (character) string

➢ VARCHAR- A variable-length non-binary string

➢ BINARY- A fixed-length binary string

➢ TEXT- A small non-binary

- **Date and Time Data Types**

➢ DATE- A date value in 'YY-MM-DD' format

➢ TIME- A time value in 'hh:mm:ss' format

➢ DATETIME- A date and time value in 'YY-MM-DD hh:mm:ss' format

➢ TIMESTAMP- A timestamp value in 'YY-MM-DD hh:mm:ss' format

➢ YEAR- A year value in YY format

MySQL -Create Table

- **Simple Table Creation:-**

#create table *table_name (*

<column_name> <data_type>[(size)] ,

<column_name> <data_type>[(size)]

);

- **Creation of Table Using SQL Constraints:**

#Create table *table_name (*

<column_name> <data_type>[(size)] <constraint> ,

<column_name> <data_type>[(size)] <constraint>

);

MySQL -Create Table

- **The various constraints that can be issued are:-**

➢ NOT NULL: - Ensures that a column cannot have null values.

➢ DEFAULT: - Provides a default value for a column when none is specified.

➢ UNIQUE: - Ensures that all values in a column are different.

➢ Primary Key: - Used to uniquely identify a row in a table.

➢ Foreign Key: - Used to ensure referential integrity of the data.

MySQL -Create Table

- **To check which table exist in current database**

  #show tables;

- **To view a table structure**

  #describe table_name;

- **To delete table**

  #drop table table_name;

MySQL –Insert Query

- **Syntax:**

    #INSERT INTO *table_name*
        VALUES (*value1*,*value2*,*value3*,...);

OR

#INSERT INTO *table_name* (*column1*,*column2*,*column3*,...)
VALUES (*value1*,*value2*,*value3*,...);

Example:

    #INSERT INTO *tutorials_tbl*
        (tutorial_title, tutorial_author, submission_date)
        VALUES ("JAVA Tutorial", "Sanjay", '2007-05-06');

MySQL –Select Query

- **Retrieve data from table**

  #SELECT *what_to_select*
  FROM *which_table*
  WHERE *conditions_to_satisfy*

  #SELECT *field1, field2,...fieldN*
    FROM *table_name1*
    [WHERE Clause]
     [LIMIT N]

  #SELECT * FROM *table_name;*

MySQL –Select Query

- **The SELECT statement is composed of several clauses:**

➢ SELECT- chooses which columns of the table you want to get the data.

➢ FROM- specifies the table from which you get the data.

➢ WHERE- filters rows to select.

➢ ORDER BY- specifies the order of the returned result set.

➢ LIMIT- constrains number of returned rows.

MySQL –Clause

- **Where Clause**

#SELECT column_name,column_name
FROM table_name
WHERE column_name operator value;

- **Order By Clause**

#SELECT column_name, column_name
FROM table_name
ORDER BY column_name ASC|DESC;

- **Like Clause**

#SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern ;

% - represents zero,one or multiple characters

_ - represents one character

MySQL –Clause

- **Distinct Clause**

#SELECT DISTINCT *column_name,column_name*
  FROM *table_name*;

- **BETWEEN Clause**

#SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name BETWEEN value1 AND value2;*

- **AND/OR Condition**

#SELECT *column_name(s)*
  FROM *table_name*
  WHERE *condition1 AND condition2 … OR condition_n;*

MySQL –Clause

- **IN Clause**

➢ **IN clause** is use to replace many OR conditions.

➢ The **IN** operator can also be used in the WHERE clause of other statements such as INSERT, UPDATE, DELETE, etc.

Syntax

#SELECT *column_list*

FROM *table_name*

WHERE *(expr|column) IN ('value1','value2',...);*

Example:

SELECT * FROM student

WHERE percent= 60 OR percent= 65 OR percent= 70 ;

SELECT * FROM student WHERE percent IN (60,65,70);

MySQL –UNION Keyword

- **UNION is used** –
- ✓ to select rows one after the other from several tables

Syntax

```
#SELECT column1, column2
UNION [DISTINCT | ALL]
SELECT column1, column2
```

Example:

```
#SELECT customerNumber id
FROM customers
UNION
SELECT employeeNumber id
FROM employees;
```

MySQL –Update/Delete

- **Update Query**

  #UPDATE *table_name*

  SET *field1=new-value1, field2=new-value2*

  [WHERE Clause]

- **Delete Query**

  #DELETE FROM *table_name* [WHERE Clause]

MySQL –Select Query

- **The SELECT statement is composed of several clauses:**

➢ GROUP BY- group rows to apply aggregate functions on each group.

➢ HAVING- filters group based on groups defined by GROUP BY clause.

```
#SELECT column_name(s)
  FROM table_name
  [WHERE column_name]
 GROUP BY column_name
  [HAVING condition];
```

MySQL – Aggregate Functions

- **Min Max Function**

  #SELECT MAX *(Field_Name)* FROM *Table_Name;*

  #SELECT MIN *(Field_Name)* FROM *Table_Name;*

- **AVG Function**

  #SELECT AVG *(Field_Name*) FROM *Table_Name;*

- **Count Function**

  #SELECT COUNT (*) FROM *Table_Name;*

- **Sum Function**

  #SELECT SUM *(column_name*) FROM *Table_Name;*

MySQL –Alter Table

- **To add a field**

  #ALTER TABLE *table_name*

  ADD *new_column_name data_type [(size)];*

- **To modify the data type of a field**

  #ALTER TABLE *table_name*

  MODIFY *column_name <new-data-type>;*

- **To delete a field**

  #ALTER TABLE *table_name*

  DROP *column_name;*

MySQL –Alter Table

- **To set a common value for a field(To set Default value)**

  #ALTER TABLE table_name ALTER Column_name SET DEFAULT value;

- **To change the name of a field**

  #ALTER TABLE *table_name*

  CHANGE <*old_Column_name*> <*new_column_name*> <*data-type*> ;

- **To change the name of a table**

  #ALTER TABLE *old_table_name*

  RENAME TO <*new_table_name* > ;

Primary Key Concept

- The **PRIMARY KEY** constraint uniquely identifies each record in a database table.
- Primary keys must contain UNIQUE values.
- A primary key column cannot contain NULL values.
- Only ONE primary key per relation.
- Primary keys typically appear as columns in relational database tables.
- A primary key column often has *AUTO_INCREMENT* attribute that generates a unique sequence for the key automatically.

Primary Key Concept

```
#CREATE TABLE users (
  user_id INT(2)  PRIMARY KEY,
  username VARCHAR(40),
  password VARCHAR(255),
  email VARCHAR(255)     );

OR
#CREATE TABLE users (
  user_id INT(2) ,
  username VARCHAR(40),
  password VARCHAR(255),
  email VARCHAR(255) ,
  PRIMARY KEY(user_id)    );
```

Primary Key Concept

- **PRIMARY KEY constraints Using Auto Increment**

```
#CREATE TABLE users (
   user_id INT AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(40),
   password VARCHAR(255),
   email VARCHAR(255)     );
```

OR

```
#CREATE TABLE roles(
   role_id INT AUTO_INCREMENT,
   role_name VARCHAR(50),
   PRIMARY KEY(role_id)     );
```

Primary Key Concept

PRIMARY KEY constraints using ALTER TABLE statement

```
#ALTER TABLE table_name
    ADD PRIMARY KEY(primary_key_column);
```

Unique Key Concept

- A **unique key** is a set of zero, one, or more attributes.
- The value(s) of these attributes are required to be unique for each tuple (row) in a relation.
- The value, or combination of values, of **unique key** attributes for any tuple should not be repeated for any other tuple in that relation.

```
#CREATE TABLE Persons
  (
  P_Id int NOT NULL,
  Name varchar(255) NOT NULL,
  UNIQUE (P_Id)
  );
```

Unique Key Concept

UNIQUE KEY constraints using ALTER TABLE statement

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
```

Foreign Key

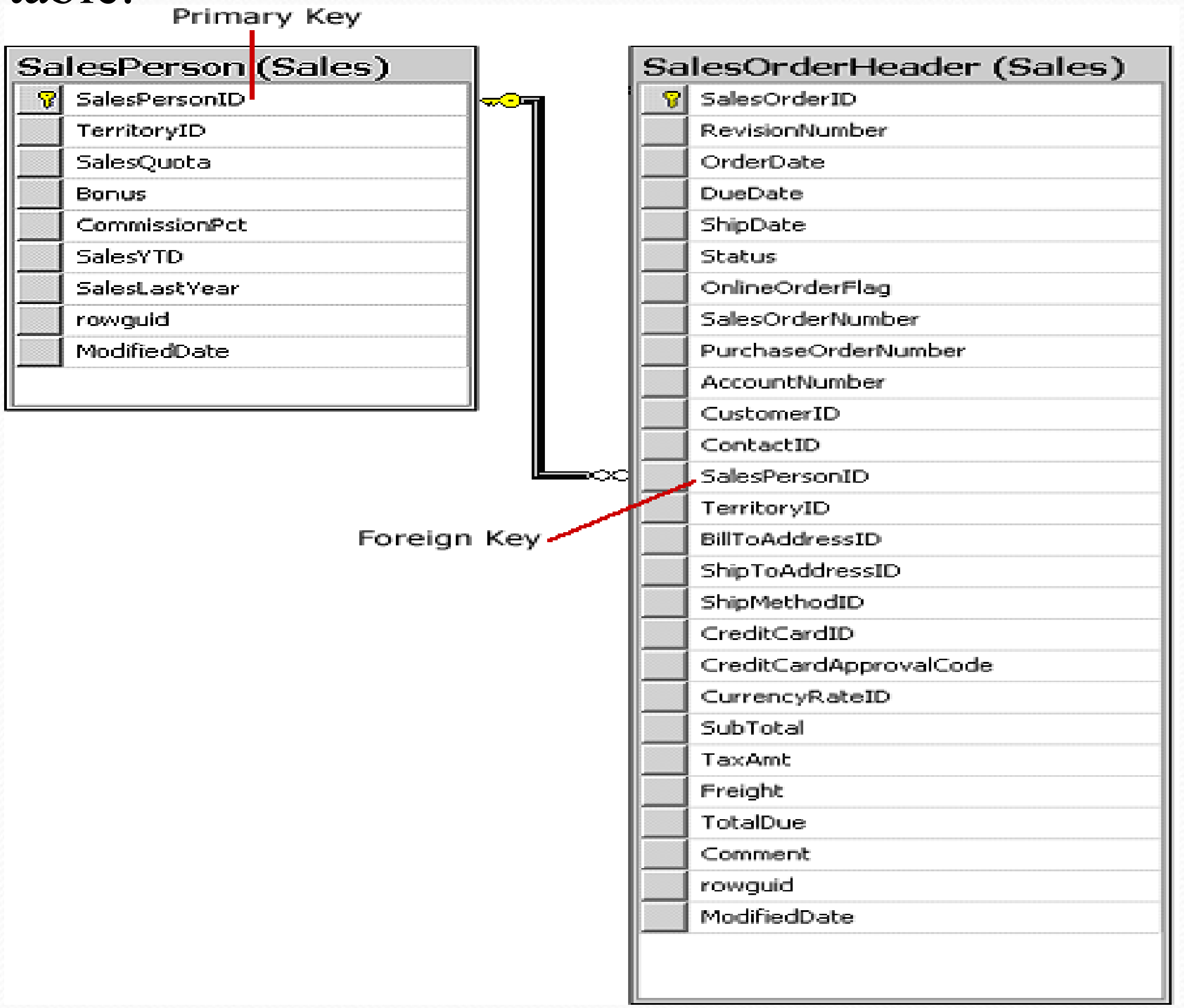
- A **FOREIGN KEY** is a field in a table that matches another field of another table. A foreign key places constraints on data in the related tables, which enables MySQL to maintain referential integrity.

- **Referential integrity** is a property of data which, when satisfied, requires every value of one attribute (column) of a relation (table) to exist as a value of another attribute in a different relation (table).

- For **referential integrity** to hold in a relational database, **any column in a base/child table that is declared a foreign key can contain either a null value, or only values from a parent table's primary key.**

Foreign Key

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.



Foreign Key

- **Foreign Key constraints using Create Statement**

```
#CREATE TABLE Persons (
    P_Id int NOT NULL,
    P_name varchar(5),
    PRIMARY KEY (P_Id) );

#CREATE TABLE Orders (
  O_Id int NOT NULL,
  OrderNo int NOT NULL,
  P_Id int,
  PRIMARY KEY (O_Id),
  FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
  );
```

Foreign Key

Foreign Key constraints using ALTER TABLE statement

```
#ALTER TABLE Table_name
    ADD FOREIGN KEY (Column_name)
    REFERENCES parent_table(columns);
```

Foreign Key Constraints :

1. **You can insert value into base/child table(FK) only if value is present in parent table(PK) or NULL**
2. **You can delete value from base/child table(FK)**
3. **You cannot delete value from parent table(PK) if value is referred in base/child table(FK)**

MySQL View

- A view is a virtual table.
- View is a data object which does not contain any data.
- Contents of the view are the resultant of a base table.
- They are operated just like base table but they don’t contain any data of their own.
- The difference between a view and a table is that views are definitions built on top of other tables (or views).
- If data is changed in the underlying table, the same change is reflected in the view.
- A view can be built on top of a single or multiple tables.

MySQL View Contd…

- **Creating View**

```
#CREATE OR REPLACE
  VIEW [view_name]
  AS
  [SELECT  statement]
```

Example:

```
#CREATE VIEW view_name AS
  SELECT column_name(s)
  FROM table_name
  WHERE condition
```

MySQL View Contd…

- **Updating Views**

```
#UPDATE view_name
        SET field1=new-value1, field2=new-value2
          [WHERE Clause]
```

Example:

```
#UPDATE TE
SET phone = 9096239923'
WHERE name = “sagar”;
```

MySQL View Contd…

- **Alter a View**

```
#ALTER [OR REPLACE]  VIEW view_name [(column_list)]
    AS
  Select_Statement;
```

- **Drop a View**

```
#DROP VIEW view name;
```

- **Display a View**

```
#SHOW CREATE VIEW view name;
```

MySQL View Contd…

- **CREATE VIEW with WHERE**

```
#CREATE VIEW view_name
 AS
      SELECT Column_name  FROM Table_name
      WHERE condition;
```

- **CREATE VIEW with AND and OR**

```
#CREATE VIEW view_name
AS
  SELECT Column_name  FROM  Table_name
  WHERE  (Condition1 AND  Condition2)
              OR  (Condition3  AND Condition4);
```

MySQL View Contd…

- **CREATE VIEW with LIKE**

```
#CREATE VIEW view_name
        AS  SELECT Column_name
         FROM Table_name
          WHERE  Column_name  LIKE ”Pattern%”
               AND Column_name NOT LIKE ”%Pattern”;
```

- **CREATE VIEW with GROUP BY**

```
#CREATE VIEW view_name
        AS  SELECT Column_name
         FROM Table_name
                GROUP BY Column_name;
```

MySQL Create User

- **CREATE USER**

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY
   'password';


# CREATE USER 'anu'@'localhost' IDENTIFIED BY 'anu';
```

- **Login using user**

```
# mysql -u anu -p
Enter password: anu
```

MySQL Grant/Revoke

- **Grant Statement**

```
#GRANT privileges ON object TO user;

#GRANT [type of permission]
        ON [database name].[table name]
        TO ‘[username]’@'localhost’;
```

Privileges

- SELECT , INSERT , UPDATE , DELETE , INDEX , CREATE , ALTER , DROP ,ALL etc.

```
#GRANT SELECT, INSERT, UPDATE, DELETE
ON mescoe.student TO 'abc'@'localhost';
```

MySQL Grant/Revoke

- **Grant Statement**

#GRANT ALL PRIVILEGES ON mescoe.student TO
‘abc'@'localhost';

#GRANT ALL PRIVILEGES ON * . * TO 'abc'@'localhost';

- **Revoke Statement**

#REVOKE privileges ON object FROM user;

MySQL Index

- A database **Index** is a data structure that improves the speed of operations in a table.
- **Indexes** can be created using one or more columns.
- Practically, indexes are also type of tables, which keep primary key or index field and a pointer to each record into the actual table.
- The users cannot see the indexes, they are just used to speed up queries and will be used by Database Search Engine to locate records very fast.
- **INSERT and UPDATE** statements take more time on tables having indexes where as **SELECT** statements become fast on those tables.

MySQL Index

- **Simple Index on Existing Table**

#CREATE INDEX [index name] ON [table name]([column name]);

- **Simple Index on New Table**

#CREATE TABLE table_name(col_name1,Col_name2, INDEX (col_name));

MySQL Index

- **Unique Index**

➢ A unique index means that two rows cannot have the same index value.

➢ Unique indexes work in much the same way as a primary key.

➢ Only one primary key, any number of unique indexes can be created with any number of fields.

```
#CREATE UNIQUE INDEX  index_name
     ON table_name ( column1, column2,...);
```

- **Index the values in a column in descending order-**

```
#CREATE UNIQUE INDEX index_name
     ON table_name (column_name DESC);
```

MySQL Index

- **ALTER command to ADD INDEX**

Unique Index

#ALTER TABLE tbl_name ADD UNIQUE index_name (col_list);

Simple Index

#ALTER TABLE tbl_name ADD INDEX index_name (col_list);

FULLTEXT index that is used for text-searching purposes

#ALTER TABLE tbl_name ADD FULLTEXT index_name (col_list);

MySQL Index

- **ALTER command to DROP INDEX**

```
# ALTER TABLE tbl_name DROP INDEX index_name;
```

Displaying INDEX Information

#SHOW INDEX FROM *table_name*;

#SHOW INDEX FROM *table_name*\G

Vertical-format output (specified by \G)

MySQL Joins

- MySQL joins are used to combine rows from two or more tables.

- **Different SQL JOINs**

➢ **INNER JOIN**: Returns all rows when there is at least one match in BOTH tables

➢ **LEFT JOIN**: Return all rows from the left table, and the matched rows from the right table

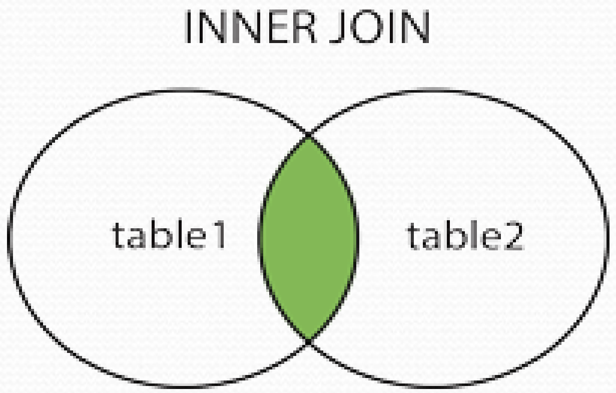
➢ **RIGHT JOIN**: Return all rows from the right table, and the matched rows from the left table

➢ **FULL JOIN**: Return all rows when there is a match or not in either left or right table

MySQL Inner Joins

- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

#SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name=table2.column_name*;



OR

SELECT *column_list*

FROM table1

INNER JOIN *table2 ON join_condition1*

INNER JOIN *table3 ON join_condition2*

...

WHERE *where_conditions;*

MySQL Inner Joins Contd…

author	count
Korth	3
Sudarshan	5
Desai	1

tcount

id	title	author	sub_date
1	DBMS	korth	2007-05-24
2	SPOS	Desai	2009-02-20
3	DBMS	Conolly	2002-01-01

tutorial

MySQL Inner Joins Contd…

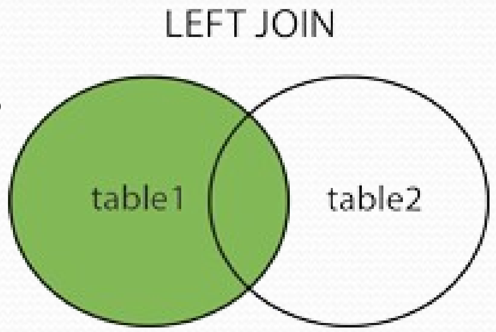
```
#SELECT tcount.author, tutorial.id, tutorial.title
    FROM tcount
    INNER JOIN tutorial
    ON tcount.author=tutorial.author;
```

author	id	title
korth	1	DBMS
desai	2	SPOS

MySQL Left Joins

- The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2).
- The result is NULL in the right side when there is no match.

```
#SELECT column_name(s)
  FROM table1
  LEFT JOIN table2
  ON table1.column_name=table2.column_name;
OR
#SELECT column_name(s)
  FROM table1
  LEFT OUTER JOIN table2
  ON table1.column_name=table2.column_name;
```



MySQL Left Joins Contd…

author	count
Korth	3
Sudarshan	5
Desai	1

tcount

id	title	author	sub_date
1	DBMS	korth	2007-05-24
2	SPOS	Desai	2009-02-20
3	DBMS	Conolly	2002-01-01

tutorial

MySQL Left Joins Contd…

```
#SELECT tcount.author, tutorial.id,tutorial.title
    FROM tcount
    LEFT JOIN tutorial
    ON tcount.author=tutorial.author;
```

author	id	title
korth	1	DBMS
sudarshan	**NULL**	**NULL**
desai	2	SPOS

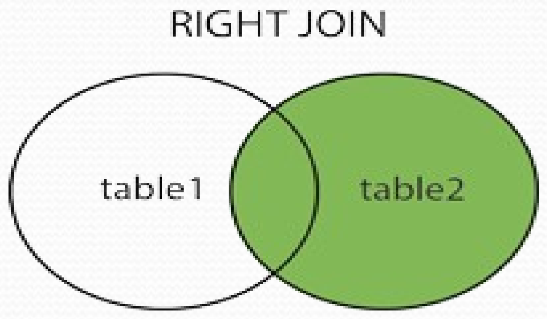
MySQL Right Joins

- The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1).
- The result is NULL in the left side when there is no match.

```
#SELECT column_name(s)
  FROM table1
  RIGHT JOIN table2
  ON table1.column_name=table2.column_name;
```

OR

```
#SELECT column_name(s)
  FROM table1
  RIGHT OUTER JOIN table2
  ON table1.column_name=table2.column_name;
```



MySQL Right Joins Contd…

author	count
Korth	3
Sudarshan	5
Desai	1

tcount

id	title	author	sub_date
1	DBMS	korth	2007-05-24
2	SPOS	Desai	2009-02-20
3	DBMS	Conolly	2002-01-01

tutorial

MySQL Right Joins Contd…

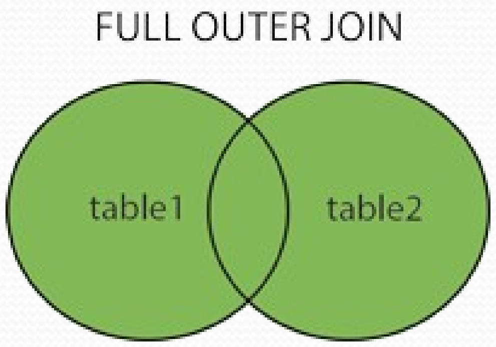
#SELECT tcount.*author, tutorial.id,tutorial.title , tutorial.author*
FROM tcount
RIGHT JOIN tutorial
ON tcount.*author=tutorial.author;*

author	id	title	author
korth	1	DBMS	korth
desai	2	SPOS	desai
NULL	3	DBMS	conolly

MySQL Full Joins

- The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).
- The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.
- UNION Keyword can be used to combine result

```
#SELECT column_name(s)
  FROM table1
  LEFT JOIN table2
  ON table1.column_name=table2.column_name
   UNION
 SELECT column_name(s)
   FROM table1
   RIGHT JOIN table2
   ON table1.column_name=table2.column_name;
```



MySQL Full Joins Contd…

author	count
Korth	3
Sudarshan	5
Desai	1

tcount

id	title	author	sub_date
1	DBMS	korth	2007-05-24
2	SPOS	Desai	2009-02-20
3	DBMS	Conolly	2002-01-01

tutorial

MySQL Full Joins Contd…

#SELECT * FROM tcount *LEFT* JOIN tutorial
ON tcount.*author=tutorial.author*

UNION

*SELECT * FROM tcount RIGHT JOIN tutorial*
ON tcount.author=tutorial.author

author	count	id	title	author	sub_date
korth	3	1	DBMS	korth	2007-05-24
sudarshan	5	**NULL**	**NULL**	**NULL**	**NULL**
desai	1	2	SPOS	desai	2009-02-20
NULL	**NULL**	3	DBMS	conolly	2002-01-01

END OF SQL