Database Management System

(18IS5DCDBM)

Unit - 3

Mrs. Bhavani K
Assistant Professor
Dept. of ISE, DSCE

DSCE
Dept. of Information Science & Engg

Unit 3

- **SQL:**
  - Overview
  - The Form of a Basic SQL Query
  - Union, Intersect and Except
  - Nested Queries
  - Aggregate Operators
  - Null Values

DSCE
Dept. of Information Science & Engg

What is SQL?

- SQL stands for Structured Query Language.
- SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems.
- SQL is a standard programming language specifically designed for storing, retrieving, managing or manipulating the data inside a relational database management system (RDBMS).
- Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. However, some features of the SQL standard are implemented differently in different database systems.

DATA TYPES

- Data types define the nature of the data that can be stored in a particular column of a table
- **Numeric Data types**
- Numeric data types are used to store numeric values. It is very important to make sure range of your data is between lower and upper boundaries of numeric data types.

INT( )	-2147483648 to 2147483647 normal 0 to 4294967295 UNSIGNED.
FLOAT	A small approximate number with a floating decimal point.
DOUBLE(,)	A large number with a floating decimal point.
DECIMAL(,)	A DOUBLE stored as a string , allowing for a fixed decimal point. Choice for storing currency values.

DSCE
Dept. of Information Science & Engg

DATA TYPES

- **Text Data Types**
- As data type category name implies these are used to store text values. Always make sure the length of the textual data do not exceed maximum lengths.

CHAR( )	A fixed section from 0 to 255 characters long.
VARCHAR( )	A variable section from 0 to 255 characters long.
BLOB	A string with a maximum length of 65535 characters.

DATA TYPES

- **Date / Time**

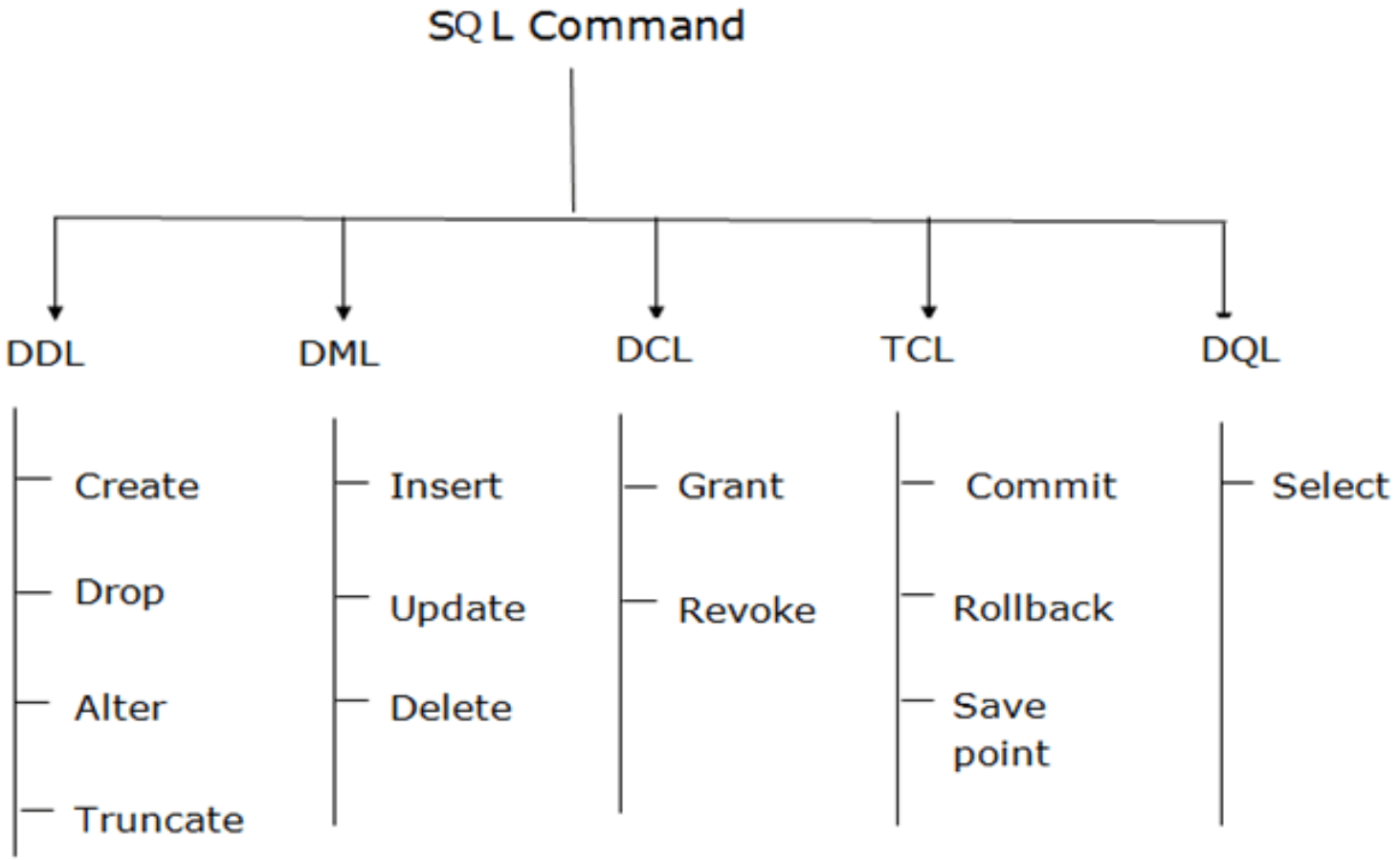
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS

SQL OVERVIEW

- The SQL language has several aspects to it.
  - **The Data Definition Language (DDL):** Consists of commands which are used to define the database
  - **The Data Manipulation Language (DML):** Consists of commands which are used to manipulate the data present in the database
  - **Data Control Language(DCL)** – Consists of commands which deal with the user permissions and controls of the database system.
  - **Transaction Control Language(TCL)** – Consist of commands which deal with the transaction of the database.
  - **Triggers and Advanced Integrity Constraints**: *triggers are actions executed by the* DBMS whenever changes to the database meet conditions specified in the trigger.
  - **Embedded and Dynamic SQL:** Embedded SQL features allow SQL code to be called from a host language such as C or COBOL. Dynamic SQL features allow a query to be constructed (and executed) at run-time.

DSCE
Dept. of Information Science & Engg

SQL Commands



DDL commands

- DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

- **Examples of DDL commands:**
  - **CREATE** – is used to create the database or its objects (like table, views, store procedure and triggers).
  - **DROP** –deletes a table in the database.
  - **ALTER**-is used to alter the structure of the database.
  - **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed. It deletes the data inside a table, but not the table itself.

DSCE
Dept. of Information Science & Engg

DDL command : CREATE DATABASE

- The basic syntax for creating a database can be given with:

  CREATE DATABASE *database_name*;

- The following SQL statement creates a database named *demo*:

  CREATE DATABASE demo;

- Creating a database does not select it for use. So, we must need to select the target database with the USE statement.
  - For example, the USE demo; command sets the *demo* database as target database for all future commands.

DDL command : CREATE TABLE

CREATE TABLE *table_name*

(*column1_name data_type constraints,*

column2_name data_type constraints, ...);

Example:

```
CREATE TABLE persons (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    birth_date DATE,
    phone VARCHAR(15) NOT NULL UNIQUE );
```

DDL command : CREATE TABLE

- There are a few additional constraints (also called *modifiers*) that are set for the table columns in the preceding statement. Constraints define rules regarding the values allowed in columns.
  - The NOT NULL constraint ensures that the field cannot accept a NULL value.
  - The PRIMARY KEY constraint marks the corresponding field as the table's primary key.
  - The AUTO_INCREMENT attribute is a MySQL extension to standard SQL, which tells MySQL to automatically assign a value to this field if it is left unspecified, by incrementing the previous value by 1. Only available for numeric fields.
  - The UNIQUE constraint ensures that each row for a column must have a unique value.

DDL commands

- **DROP:** It is used to delete both the structure and record stored in the table.
  - **Syntax**
    - DROP TABLE ;
  - **Example**
    - DROP TABLE EMPLOYEE;

- **TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.
  - **Syntax:**
    - TRUNCATE TABLE table_name;
  - **Example:**
    - TRUNCATE TABLE EMPLOYEE;

DSCE
Dept. of Information Science & Engg

DDL commands

- **ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.
- **Adding New Columns**

  Syntax:

  ALTER TABLE <table_name>

  ADD (<NewColumnName> <Data_Type>(<size>),.....n)

  Example:

  ALTER TABLE Student ADD (Age number(2), Marks number(3));

- **Dropping a Column from the Table**

  Syntax:

  ALTER TABLE <table_name> DROP COLUMN <column_name>

  Example:

  ALTER TABLE Student DROP COLUMN Age;

- **To modify existing column in the table:**

  ALTER TABLE MODIFY(COLUMN DEFINITION....);

  ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

- **Modifying Existing Table**

  Syntax:

  ALTER TABLE <table_name> MODIFY (<column_name> <NewDataType>(<NewSize>))

  Example:

  ALTER TABLE Student MODIFY (Name Varchar2(40));

DML(Data Manipulation Language)

- The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

- **Examples of DML:**
  - **INSERT** – is used to insert data into a table.
  - **UPDATE** – is used to update existing data within a table.
  - **DELETE** – is used to delete records from a database table.

DML commands

- **INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.
  - **Syntax:**

```
INSERT INTO TABLE_NAME
(col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
Or
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
```

  - **For example:**

```
INSERT INTO Book (Author, Subject) VALUES (“Navathe", "DBMS");
```

DSCE
Dept. of Information Science & Engg

DML commands

- **UPDATE:** This command is used to update or modify the value of a column in the table.
  - **Syntax:**
    - UPDATE table_name SET [column_name1= value1,...column_name N = valueN] [WHERE CONDITION]
  - **For example:**

```
UPDATE students
SET User_Name = 'Ram'
WHERE Student_Id = 3
```

- **DELETE:** It is used to remove one or more row from a table.
  - **Syntax:**
    - DELETE FROM table_name [WHERE condition];
  - **For example:**

```
DELETE FROM Book
WHERE Author="Navathe";
```

DCL(Data Control Language)

- DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

- **Examples of DCL commands:**
  - **GRANT**-gives user's access privileges to database.
  - **REVOKE**-withdraw user's access privileges given by using the GRANT command.

DCL commands

- **Grant:** It is used to give user access privileges to a database.
  - **Example**

    GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

- **Revoke:** It is used to take back permissions from the user.
  - **Example**

    REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

TCL(transaction Control Language)

- TCL commands deals with the transaction within the database.

- **Examples of TCL commands:**
  - **COMMIT**– commits a Transaction.
  - **ROLLBACK**– rollbacks a transaction in case of any error occurs.
  - **SAVEPOINT**–sets a savepoint within a transaction.
  - **SET TRANSACTION**–specify characteristics for the transaction.

TCL Commands

- **Commit:** Commit command is used to save all the transactions to the database.
  - **Syntax:**

    COMMIT;
  - **Example:**

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

- **Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.
  - **Syntax:**

    ROLLBACK;
  - **Example:**

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

- **SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.
  - **Syntax:**

    SAVEPOINT SAVEPOINT_NAME;

DQL(Data Query Language)

- Data query language is used to fetch data from the database.
- It uses only one command:
  - SELECT
    - It is used to select the attribute based on the condition described by WHERE clause.

Example Schemas

- A number of sample queries will be presented using the following table definitions:

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)

Boats(*bid:* integer, *bname:* string, *color:* string)

Reserves (*sid:* integer, *bid:* integer, *day:* date)

Example States

sid	sname	rating	age
22	Dustin	16	45
31	lubber	8	55.5
58	rusty	10	35
95	Bob	20	63.5

An Instance of sailors

bid	bname	color
101	Interlake	blue
102	Interlake	green
103	Clipper	red
104	Marine	red

An Instance of Boats

sid	bid	day
22	101	1996-10-10T00:00:00Z
58	103	1996-12-11T00:00:00Z
58	104	1996-12-11T00:00:00Z

An Instance *R2* of reserves

THE FORM OF A BASIC SQL QUERY

- The basic form of an SQL query is as follows:

  SELECT [DISTINCT] select-list
  FROM from-list
  WHERE qualification/condition

- Every query must have a SELECT clause which specifies columns to be retained in the result and
- A FROM clause which specifies table or a cross-product of tables
- The optional WHERE clause specifies selection conditions on the tables mentioned in the FROM clause.

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *from-list.*
  - Discard resulting tuples if they fail *qualifications.*
  - Delete attributes that are not in *select-list.*
  - If DISTINCT is specified, eliminate duplicate rows.

- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers.*

Example

- Find the' names of all sailors

  SELECT distinct S.sname

  FROM Sailors S

sname
Dustin
lubber
rusty
Bob

Answer with distinct

sname
Dustin
Dustin
lubber
rusty
Bob

Answer without distinct

Example

- Find the names of sailors 'Who have reserved boat number 103.

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid=103
```

sname
rusty

Answer

sid	sname	rating	age
22	Dustin	16	45
28	Dustin	7	50.5
31	lubber	8	55.5
58	rusty	10	35
95	Bob	20	63.5

Relation Sailors

sid	bid	day
22	101	1996-10-10T00:00:00Z
58	103	1996-12-11T00:00:00Z
58	104	1996-12-11T00:00:00Z

Relation Reserves

A Note on Range Variables

- Really needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

```
SELECT S.sname
FROM Sailors AS S, Reserves AS R
WHERE S.sid=R.sid AND bid=103;
```

Optional

OR

```
SELECT sname
FROM Sailors, Reserves
WHERE Sailors.sid=Reserves.sid
AND bid=103;
```

It is good style, however, to use range variables always!

Examples

- *Find the sids of sailors who have reserved a red boat.*

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE B.bid = R.bid AND B.color = 'red'
```

sid
58
58

Answer

bid	bname	color
101	Interlake	blue
102	Interlake	green
103	Clipper	red
104	Marine	red

Relation Boats

sid	bid	day
22	101	1996-10-10T00:00:00Z
58	103	1996-12-11T00:00:00Z
58	104	1996-12-11T00:00:00Z

Relation Reserves

Example

- *Find the names of sailors Who have reserved a red boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
```

sname
rusty
rusty

Answer

sid	sname	rating	age
22	Dustin	16	45
28	Dustin	7	50.5
31	lubber	8	55.5
58	rusty	10	35
95	Bob	20	63.5

Relation Sailors

bid	bname	color
101	Interlake	blue
102	Interlake	green
103	Clipper	red
104	Marine	red

Relation Boats

sid	bid	day
22	101	1996-10-10T00:00:00Z
58	103	1996-12-11T00:00:00Z
58	104	1996-12-11T00:00:00Z

Relation Reserves

Example

- Find names of sailors who've reserved at least one boat

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid;
```

sid	sname	rating	age
22	Dustin	16	45
28	Dustin	7	50.5
31	lubber	8	55.5
58	rusty	10	35
95	Bob	20	63.5

Relation Sailors

sid	bid	day
22	101	1996-10-10T00:00:00Z
58	103	1996-12-11T00:00:00Z
58	104	1996-12-11T00:00:00Z

Relation Reserves

sname
Dustin
rusty
rusty

Answer

Expressions and Strings in select-list

- Select-list: ***expression AS column_name***
  - ***expression: any arithmetic or string expression*** over column names and constants.
  - ***column_name: a new name for this column in the*** output of the query.
- Qualification: *expression1 = expression2*

Example

- Compute increments for the ratings of persons who have sailed two different boats **on the same day.**

*who have sailed two different boats **on the same day? Please show their sailor ids.***

```
SELECT DISTINCT R1.sid
FROM Reserves R1, Reserves R2
WHERE R1.sid=R2.sid
AND R1.day = R2.day AND
R1.bid<>R2.bid;
```

sid
58

Names of persons who have sailed two different boats **on the same day**.

```
SELECT distinct S.sname
FROM Sailors S, Reserves R1, Reserves R2
WHERE S.sid=R1.sid AND R1.sid=R2.sid
AND R1.day = R2.day AND R1.bid<>R2.bid;
```

sname
rusty

Example contd…

- Compute increments for the ratings of persons who have sailed two different boats **on the same day**.

```
SELECT distinct S.sname, S.rating+1 AS rating
FROM Sailors S, Reserves R1, Reserves R2
WHERE S.sid=R1.sid AND S.sid=R2.sid
AND R1.day = R2.day AND R1.bid<>R2.bid;
```

sname	rating
rusty	11

Example

- Find names of sailors whose ratings are different in the given way i.e. one rating is twice the other

```
SELECT S1.sname AS name1, S2.sname AS name2
FROM Sailors S1, Sailors S2
WHERE 2*S1.rating = S2.rating;
```

sid	sname	rating	age
22	Dustin	16	45
28	Dustin	7	50.5
31	lubber	8	55.5
58	rusty	10	35
95	Bob	20	63.5

Relation Sailors

name1	name2
lubber	Dustin
rusty	Bob

Answer

Expressions and Strings in select-list

- AS can be used to name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.
- Find doubles (of ages of sailors and one field defined by expression) for sailors whose names begin and end with B and contain at least three characters.

```
SELECT S.age, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

age	age2
63.5	127

Answer

sid	sname	rating	age
22	Dustin	16	45
28	Dustin	7	50.5
31	lubber	8	55.5
58	rusty	10	35
95	Bob	20	63.5

Relation Sailors

Union

- UNION/OR: Can be used to compute the union of any two *union-compatible sets of tuples.*

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid
AND (B.color='red' OR B.color='green');
```

sid
58
58

```
SELECT R1.sid
FROM Boats B1, Reserves R1
WHERE R1.bid=B1.bid AND B1.color='red'
UNION
SELECT R2.sid
FROM Boats B2, Reserves R2
WHERE R2.bid=B2.bid
AND B2.color='green'
```

sid
58

Example

- Find all sids' of sailors who've a rating of 10 or reserved boat 104.

```
SELECT S.sid
FROM Sailors S
WHERE S.rating =10
UNION
SELECT R.sid
FROM Reserves R
WHERE R.bid = 101
```

sid
58
22

Intersect

- INTERSECT/AND: Can be used to compute the intersection of any two *intersect-compatible sets of* tuples.

- Find sid's of sailors who've reserved a red and a green boat

```
SELECT R1.sid
FROM Reserves R1, Reserves R2, Boats B1, Boats B2 WHERE R1.sid = R2.sid and
R1.bid=B1.bid AND R2.bid=B2.bid AND (B1.color='red' AND B2.color='green');
```

sid
58

```
SELECT R1.sid
FROM Boats B1, Reserves R1
WHERE R1.bid=B1.bid AND B1.color='red'
INTERSECT
SELECT R2.sid
FROM Boats B2, Reserves R2
WHERE R2.bid=B2.bid AND B2.color='green'
```

Not supported in mysql

Except (Difference)

- EXCEPT: Can be used to compute the set-difference of any two *except-compatible sets of tuples.*

- Find sid's of sailors who've reserved red boats but not green boats.

```
SELECT R1.sid
FROM Boats B1, Reserves R1
WHERE R1.bid=B1.bid AND B1.color='red'
EXCEPT
SELECT R2.sid
FROM Boats B2, Reserves R2
WHERE R2.bid=B2.bid AND B2.color='green'
```

Not supported in mysql

UNION, INTERSECT, and EXCEPT

- The default for UNION queries is that duplicates *are* eliminated!
  - To retain duplicates, UNION ALL must be used;
    - the number of copies of a row in the result is always m + *n, where m and n are the numbers* of times that the row appears in the two parts of the union.

- Similarly, INTERSECT ALL retains duplicates
  - the number of copies of a row in the result is *min(m, n)-and*

- *EXCEPT ALL also retains duplicates*
  - *the number of copies* of a row in the result is m - *n, where 'm corresponds to the first relation.*

DSCE
Dept. of Information Science & Engg

Nested Queries

- One of the most powerful features of SQL is nested queries.
- A nested query is a query that has another query embedded within it; the embedded query is called a subquery.
  - The embedded query can be a nested query itself; thus queries that have very deeply nested structures are possible.
- When writing a query, we sometimes need to express a condition that refers to a table that must itself be computed. The query used to compute this subsidiary table is a subquery and appears as part of the main query.
  - A subquery typically appears within the WHERE clause of a query.
  - Subqueries can sometimes appear in the FROM clause or the HAVING clause

Nested Queries

- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
- A 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.
- There are mainly two types of nested queries:
  - **Independent Nested Queries:** In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.
  - **Co-related Nested Queries:** In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

DSCE
Dept. of Information Science & Engg

Rules for Nested Queries

- There are a few rules that subqueries must follow
  - Subqueries must be enclosed within parentheses.
  - A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
  - An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
  - Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
  - The SELECT list cannot include any references to values that evaluate to a BLOB, CLOB.
  - A subquery cannot be immediately enclosed in a set function.
  - The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Example

- Consider the CUSTOMERS table having the following records

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	25	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bangalore	8500
6	Komal	22	MP	4500
7	Madhu	24	Indore	10000

Subqueries with the SELECT 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

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

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bangalore	8500
7	Madhu	24	Indore	10000

DSCE
Dept. of Information Science & Engg

Subqueries with the INSERT Statement

- Subqueries also can be used with INSERT statements.
- The INSERT statement uses the data returned from the subquery to insert into another table.
  - The selected data in the subquery can be modified with any of the character, date or number functions.
- The basic syntax is as follows.

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
SELECT [ *|column1 [, column2 ]
FROM table1 [, table2 ]
[ WHERE VALUE OPERATOR ]
```

Example

- Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

```
Create table CUSTOMERS_BKP
(ID integer Primary key,
NAME varchar(15),
AGE integer,
ADDRESS varchar(15),
SALARY real
);
```

```
INSERT INTO CUSTOMERS_BKP
SELECT * FROM CUSTOMERS
WHERE ID IN (SELECT ID FROM
CUSTOMERS) ;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	25	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bangalore	8500
6	Komal	22	MP	4500
7	Madhu	24	Indore	10000

Subqueries with the UPDATE Statement

- The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

- The basic syntax is as follows.

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

Example

- Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
UPDATE CUSTOMERS
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
  WHERE AGE >= 27 );
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	25	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bangalore	8500
6	Komal	22	MP	4500
7	Madhu	24	Indore	10000

Customer

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	25	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bangalore	8500
6	Komal	22	MP	4500
7	Madhu	24	Indore	10000

Customers_BKP

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	125.0
2	Khilan	25	Delhi	1500
3	kaushik	25	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bangalore	2125
6	Komal	22	MP	4500
7	Madhu	24	Indore	10000

Updated Table

Subqueries with the DELETE Statement

- The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

- The basic syntax is as follows:

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

Example

- Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
DELETE FROM CUSTOMERS
  WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
    WHERE AGE >= 27 );
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	25	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bangalore	8500
6	Komal	22	MP	4500
7	Madhu	24	Indore	10000

Customer

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	25	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bangalore	8500
6	Komal	22	MP	4500
7	Madhu	24	Indore	10000

Customers_BKP

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500
3	kaushik	25	Kota	2000
4	Chaitali	25	Mumbai	6500
6	Komal	22	MP	4500
7	Madhu	24	Indore	10000

Answer

The SQL ANY and ALL Operators

- The ANY and ALL operators are used with a WHERE or HAVING clause.
- The ANY operator returns true if any of the subquery values meet the condition.
- The ALL operator returns true if all of the subquery values meet the condition.
- Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY/ALL
(SELECT column_name FROM table_name WHERE condition);
```

Example

Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Orders:

order_id	c_id	order_date	purch_amt
1	407	2017-03-03	1000
2	405	2017-03-05	5000
3	408	2017-01-18	1500
4	404	2017-02-05	3000

Example: Any operator

- Find all orders with an amount larger than any amount for a customer from website jkl.com.

```
SELECT *
FROM orders
WHERE purch_amt > ANY
  (SELECT purch_amt
   FROM orders a, customers b
   WHERE  a.c_id=b.customer_id
   AND b.website='jkl.com');
```

Output:

order_id	c_id	order_date	purch_amt
2	405	2017-03-05	5000

Example: All operator

- Find all orders with an amount larger than all amount for a customer from website jkl.com.

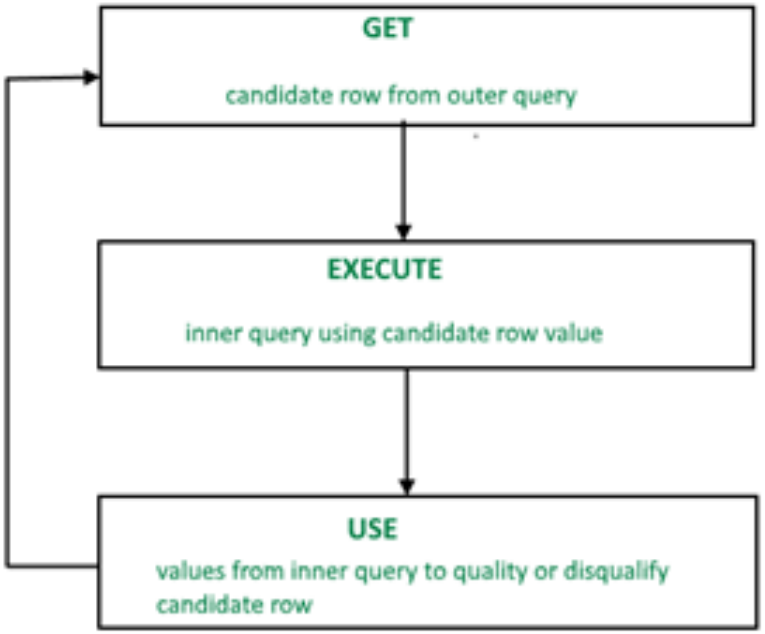
```
SELECT *
FROM orders
WHERE purch_amt > All
  (SELECT purch_amt
    FROM orders a, customers b
    WHERE  a.c_id=b.customer_id
    AND b.website='jkl.com');
```

Output:

order_id	c_id	order_date	purch_amt
2	405	2017-03-05	5000

Correlated Subqueries

- Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



DSCE
Dept. of Information Science & Engg

Nested Subqueries Versus Correlated Subqueries

- With a normal nested subquery, the inner **SELECT** query runs first and executes once, returning values to be used by the main query.

- A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

The SQL EXISTS / NOT EXISTS Operator

- The EXISTS operator is used to test for the existence of any record in a subquery.
  - The EXISTS operator returns true if the subquery returns one or more records.
- The NOT EXISTS operator is used to test for the non-existence of any record in a subquery.
  - The NOT EXISTS operator returns true if the subquery doesnot return any records.
- It can be used in a SELECT, UPDATE, INSERT or DELETE statement.
- Syntax

```
SELECT column_name(s)
 FROM table_name
 WHERE [NOT ]EXISTS
 (SELECT column_name FROM table_name WHERE condition);
```

Example

- Consider the following two relation “Customers” and “Orders”.

Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05

Example

- Fetch the first and last name of the customers who placed atleast one order.

```
SELECT fname, lname
FROM Customers
WHERE EXISTS (SELECT *
              FROM Orders
              WHERE Customers.customer_id = Orders.c_id);
```

Output:

fname	lname
Shubham	Gupta
Divya	Walecha
Rajiv	Mehta
Anand	Mehra

Example

- Fetch last and first name of the customers who has not placed any order.

```
SELECT lname, fname
FROM Customer
WHERE NOT EXISTS (SELECT *
                  FROM Orders
                  WHERE Customers.customer_id = Orders.c_id);
```

Output:

lname	fname
Singh	Dolly
Chauhan	Anuj
Kumar	Niteesh
Jain	Sandeep

DSCE
Dept. of Information Science & Engg

Example : **Using EXISTS condition with DELETE statement**

- Delete the record of all the customer from Order Table whose last name is ‘Mehra’.

```
DELETE
FROM Orders
WHERE EXISTS (SELECT *
                FROM customers
                WHERE Customers.customer_id = Orders.cid
                AND Customers.lname = 'Mehra');
```

```
SELECT * FROM Orders;
```

Output:

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
4	404	2017-02-05

Using EXISTS condition with UPDATE statement

- Update the lname as ‘Kumari’ of customer in Customer Table whose customer_id is 401.

```
UPDATE Customers
SET lname = 'Kumari'
WHERE EXISTS (SELECT *
              FROM Customers
              WHERE customer_id = 401);
```

```
SELECT * FROM Customers;
```

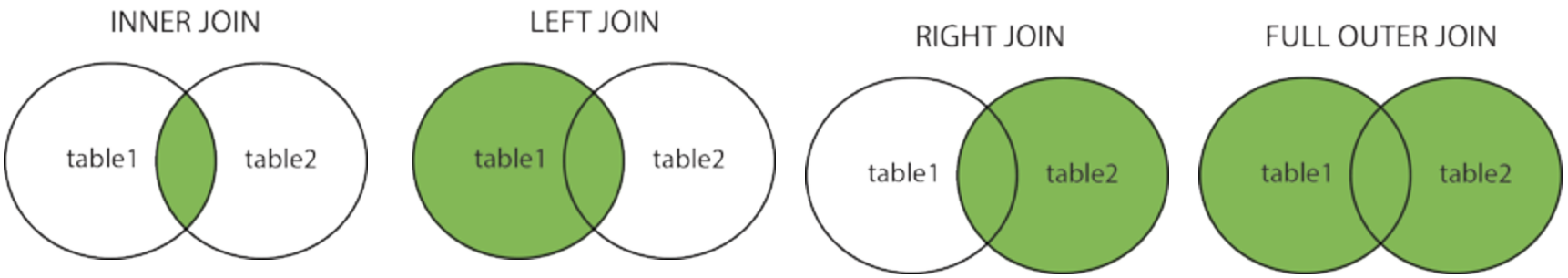
Output:

customer_id	lname	fname	website
401	Kumari	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

SQL JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

- Different Types of SQL JOINs
  - **(INNER) JOIN**: Returns records that have matching values in both tables
  - **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
  - **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
  - **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



SQL INNER JOIN Example

- The INNER JOIN command returns rows that have matching values in both tables.

- The following SQL statement selects all orders with customer information:

```
SELECT Orders.order_id, Customers.fname,Customers.lname
FROM Orders
INNER JOIN Customers ON Orders.c_id = Customers.customer_id;
```

order_id	fname	lname
1	Rajiv	Mehta
2	Divya	Walecha
3	Anand	Mehra
4	Shubham	Gupta

LEFT JOIN

- The LEFT JOIN command returns all rows from the left table, and the matching rows from the right table. The result is NULL from the right side, if there is no match.

- The following SQL will select all customers, and any orders they might have:

```
SELECT Customers.fname,Customers.lname, Orders.order_id
FROM Customers
LEFT JOIN Orders ON Orders.c_id = Customers.customer_id
ORDER BY Customers.customer_id;
```

fname	lname	order_id
Dolly	Singh	(null)
Anuj	Chauhan	(null)
Niteesh	Kumar	(null)
Shubham	Gupta	4
Divya	Walecha	2
Sandeep	Jain	(null)
Rajiv	Mehta	1
Anand	Mehra	3

RIGHT JOIN

- The RIGHT JOIN command returns all rows from the right table, and the matching records from the left table. The result is NULL from the left side, when there is no match.

- The following SQL will return all employees, and any orders they might have placed:

```
SELECT Orders.order_id, Customers.fname,Customers.lname
FROM Orders
RIGHT JOIN Customers ON Orders.c_id = Customers.customer_id
ORDER BY Orders.order_id;
```

order_id	fname	lname
(null)	Anuj	Chauhan
(null)	Niteesh	Kumar
(null)	Dolly	Singh
(null)	Sandeep	Jain
1	Rajiv	Mehta
2	Divya	Walecha
3	Anand	Mehra
4	Shubham	Gupta

FULL OUTER JOIN

- The FULL OUTER JOIN command returns all rows when there is a match in either left table or right table.

- The following SQL statement selects all customers, and all orders:

```
SELECT Customers.fname,Customers.lname, Orders.order_id
FROM Customers
FULL OUTER JOIN Orders ON Orders.c_id = Customers.customer_id
ORDER BY Customers.customer_id;
```

Not supported in mysql

DSCE
Dept. of Information Science & Engg

AGGREGATE FUNCTIONS

- Used to accumulate information from multiple tuples, forming a single-tuple summary

- Built-in aggregate functions
  1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
  2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
  3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
  4. MAX (A): The maximum value in the A column.
  5. MIN (A): The minimum value in the A column.

- Used in the SELECT clause

- Examples:
  - *How many movies were directed by Steven Spielberg?*

```
SELECT COUNT(*)
FROM Film
WHERE director='Steven Spielberg‘;
```

    - All tuples in result are counted, *with duplicates!*
    - COUNT(title) or COUNT(director) give same result!
    - COUNT(DISTINCT year) would include each year only once!

- *What was the total movie profit since 2010, across how many directors?*

```
SELECT SUM(gross - budget), COUNT(DISTINCT director)
FROM Film
WHERE year >= 2010;
```

GROUPING BEFORE AGGREGATION

- How can we answer a query such as

  "How many films were directed by each director after 2001?"

  •Need to produce a result with one tuple per director

  1.Partition relation into subsets of tuples based on **grouping column(s)**

  2.Apply function to each such group independently

  3.Produce one tuple per group

- **GROUP BY clause to specify grouping attributes**

```
SELECT director, COUNT(*)
FROM Film
WHERE year > 2001
GROUP BY director;
```

- Every selector in SELECT clause must be a grouping column or an aggregation function
  - e.g., SELECT director, year, COUNT(*)

    would not be allowed unless *also grouping by year*

    i.e., GROUP BY director, year

DSCE
Dept. of Information Science & Engg

HAVING CLAUSE

- After partitioning into groups, whole partitions can be discarded.
  - Provides a condition on the grouped tuples

```
SELECT    Dname, COUNT (*)
FROM      DEPARTMENT, EMPLOYEE
WHERE     Dnumber=Dno AND Salary>40000
GROUP BY  Dname
HAVING    COUNT (*) > 5;
```

- Having clause cannot reference individual tuples within group
  - Can reference grouping column(s) and aggregates only

- Contrast WHERE clause to HAVING clause

Note: As for aggregation, no GROUP BY clause means relation treated as one group

DSCE
Dept. of Information Science & Engg

ORDERING OF QUERY RESULTS

- Final output of a query can be sorted by one or more column values
- Use **ORDER BY clause**
  - Keyword **DESC for descending order of values**
  - Optionally use keyword **ASC for ascending order (default)**

- Example

```
SELECT dept, term,
COUNT(DISTINCT instructor) AS num_instructors
FROM Course
GROUP BY dept, term;
ORDER BY dept, term DESC;
```

Course

dept	cnum	instructor	term

- Note that this is sorted ascending by department.
- Within each department, terms sorted in descending order.
- What if DISTINCT omitted? What if term omitted from SELECT clause? What if dept omitted from GROUP BY clause? What if dept omitted from ORDER BY clause?

Queries With GROUP BY and HAVING

SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification

- target-list contains
  (i) attribute names
  (ii) terms with aggregate operations (e.g., MIN (S.age)).

- Attributes used in target-list must be in grouping-list.
  - Each answer tuple corresponds to a group, and these attributes must have a single value per group. (A group is a set of tuples that have the same value for all attributes in grouping-list.)

DSCE
Dept. of Information Science & Engg

Conceptual Evaluation

1. Compute cross-product of relation-list.
2. Discard tuples that fail qualification.
3. Delete `unnecessary' fields.
4. Partition remaining tuples into groups by the value of attributes in grouping-list.
5. Apply group-qualification to eliminate some groups.
   - Expressions in group-qualification must have a single value per group.
   - Attribute in group-qualification that is not an argument of an aggregate op also appears in grouping-list. (SQL does not exploit primary key semantics here!)

One answer tuple is generated per qualifying group.

Find age of youngest sailor with age>=18 for each rating with at least 2 such sailors.

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) >= 2
```

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

Find age of youngest sailor with age>=18 for each rating with at least 2 such sailors

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

rating	minage
3	25.5
7	35.0
8	25.5

Note: irrelevant attributes omitted on this and following slides.

Find age of the youngest sailor with age>=18, for each rating with at least 2 such sailors and with every sailor under 60.

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

rating	minage
7	35.0
8	25.5

What is the result of changing EVERY to ANY?

HAVING COUNT (*) >= 2 AND EVERY (S.age <=60)

Find age of the youngest sailor with age>=18, for each rating with at least 2 sailors between 18 and 60.

```
SELECT  S.rating,  MIN (S.age)
                          AS minage
FROM  Sailors S
WHERE  S.age >= 18 AND S.age <= 60
GROUP BY  S.rating
HAVING  COUNT (*) >= 2
```

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

SUMMARY OF SQL QUERIES

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];

1.Assemble all tables according to **From clause (“,” means to use X).**
2.Keep only tuples matching **Where clause.**
3.Group into blocks based on **Group By clause.**
4.Keep only blocks matching **Having clause.**
5.Create one tuple for each block using **Select clause.**
6.Order resulting tuples according to **Order By clause.**

NULL VALUES

- Null
  - A special value that denote **unknown** (e.g., a rating has not been assigned) or **inapplicable** (e.g., no spouse's name)
  - The presence of null complicates many issues
    - Special operators are needed to check if value is/is not null
    - Is (rating>8) true or false when rating is null? What about AND, OR and NOT connectives?
      - We need a 3-valued logic (true, false and unknown)
    - Meaning of SQL constructs must be defined carefully (e.g., WHERE clause eliminates rows that don't evaluate to true)
    - New operators (in particular, outer joins) are possible/needed

DSCE
Dept. of Information Science & Engg

Working with NULL

- NULL op constant evaluates to unknown
  - op is one of <, >, =, <=, >=, =
  - What about NULL = NULL?
- NOT unknown evaluates to unknown
- true OR unknown evaluates to true
  - What about false OR unknown?
- false AND unknown evaluates to false
- Definition of a duplicate: corresponding columns are either equal or both
- have value NULL
  - Implicitly evaluates (NULL = NULL) as true
- Arithmetic operators (+, -, *, /) return NULL if any input is NULL
- Aggregate operators affected differently
  - COUNT(*) not affected
  - All others, including COUNT(column), discard NULL values before computing the aggregate
    - Compare result of SUM(column) to using + on the same set of values
    - What if all values in the column are NULL?

Thank you