DATABASE MANAGEMENT SYSTEM

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Basics of SQL

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the below resource and persons:
- Author slides from “Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

DATABASE MANAGEMENT SYSTEM

Unit 3 : SQL

PES UNIVERSITY

1. SQL datatypes and Advanced data types like Blob Clob
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

PES UNIVERSITY

T1 : CHAPTER 6.1 Basic SQL

R1: CHAPTER 4.5 Advanced Data Types like CLOB, BLOB

Overview

- SQL Data Definition and Data Types
- Advanced Data Types like CLOB, BLOB

Introduction to SQL Commands

- SQL language
  - Considered one of the major reasons for the commercial success of relational databases
- SQL
  - SQL Actually comes from the word “SEQUEL” which was the original term used in the paper: “SEQUEL TO SQUARE” by Chamberlin and Boyce. IBM could not copyright that term, so they abbreviated to SQL and copyrighted the term SQL.
  - Now popularly known as “Structured Query language”.
  - SQL is a practical rendering of the relational data model with syntax

SQL Data Definition, Data Types, Standards

- Terminology:
  - **Table**, **row**, and **column** used for relational model terms relation, tuple, and attribute
- `CREATE` statement
  - Main SQL command for data definition
- The language has features for : Data definition, Data Manipulation, Transaction control ,Indexing ,Security specification (Grant and Revoke), Active databases, Multi-media , Distributed databases etc.

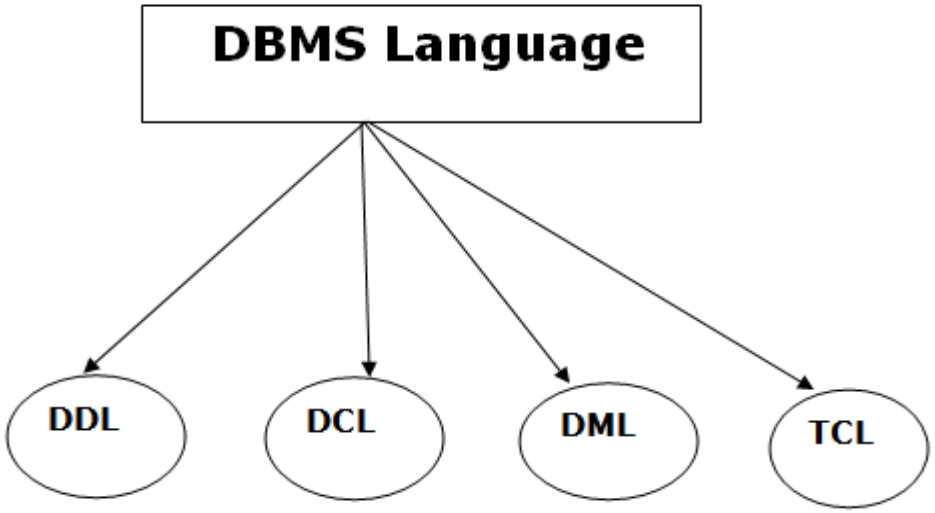
Schema Concepts in SQL

- **SQL schema**
  - Identified by a **schema name**
  - Includes an **authorization identifier** and **descriptors** for each element
- **Schema elements** include
  - Tables, constraints, views, domains, and other constructs
- Each statement in SQL ends with a **semicolon**

- `CREATE SCHEMA` statement
  - **`CREATE SCHEMA Lib AUTHORIZATION 'Jsmith';`**
- **Catalog**
  - Named collection of schemas in an SQL environment
- SQL also has the concept of a cluster of catalogs.

Database Management Systems

Types of Database Languages



Data Definition Language (DDL) statements are used to define the database structure or schema.

Data Manipulation Language (DML) statements are used for managing data within schema objects

DCL is the abstract of Data Control Language. Data Control Language includes commands such as GRANT, and is concerned with rights, permissions, and other controls of the database system.

Transaction Control Language (TCL) is used to run the changes made by the DML statement.

Types of Database Languages: DDL

DDL includes commands such as CREATE, ALTER, and DROP statements. DDL is used to CREATE, ALTER, OR DROP the database objects (Table, Views, Users).

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

Types of Database Languages: DDL

CREATE TABLE

Syntax:

```
CREATE TABLE table_name(
Col_name1 datatype(),
Col_name2 datatype(),…
Col_namen datatype(),
);
```

Example:

```
CREATE TABLE DDL  (
 id int,
 DDL_Type varchar(50),
 DDL_Value int
  );
```

Types of Database Languages: DDL

B) ALTER TABLE

1) ADD

Syntax:

```
ALTER TABLE table_name
ADD Col_name datatype()...;
```

2) MODIFY

Syntax:

```
ALTER TABLE table_name
MODIFY (fieldname datatype()...);
```

3) RENAME

Syntax:

```
ALTER TABLE table_name
RENAME COLUMN (Old_fieldname TO New_fieldname...);
```

4) DROP

Syntax:

```
ALTER TABLE table_name DROP COLUMN
column_name;
```

Types of Database Languages: DDL

C) DESCRIBE TABLE

Syntax:

DESCRIBE TABLE NAME;

D) DROP TABLE

Syntax:

DROP Table name; // Complete table structure will be dropped

E) RENAME

Rename a table

Syntax:

RENAME table table_name to new table_name

F) TRUNCATE

Syntax:

TRUNCATE TABLE table_name; // delete complete data from an existing table. Table Structure remains

Types of Database Languages: DML

Data Manipulation Language (DML) statements are used for managing data within schema objects

DML deals with data manipulation, and therefore includes most common SQL statements such as SELECT, INSERT, etc. DML allows adding / modifying / deleting data itself.

DML Commands

1.INSERT

2.SELECT

3.UPDATE

4.DELETE

Types of Database Languages: DML

1) INSERT

Syntax-1:

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN) VALUES (value1, value2, value3,...valueN);

Syntax-2

INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);

2) SELECT

Syntax:

SELECT * FROM <table_name>;

SELECT column1, column2, columnN FROM table_name;

3) UPDATE

Syntax:

UPDATE table_name SET column1 = value1, column2 = value2...., columnN = valueN WHERE [condition];

4) DELETE

Syntax:

DELETE FROM table_name WHERE [condition];

Types of Database Languages: DCL & TCL

Data Control Language includes commands such as GRANT, and is concerned with rights, permissions, and other controls of the database system.

1) GRANT

It provides the user's access privileges to the database.

Syntax:

GRANT privilege_name on object_name to {user_name | public | role_name};

Privilege_name are SELECT,UPDATE,DELETE,INSERT,ALTER,ALL

2) Revoke :

Revoke command withdraw user privileges on database objects if any granted.

Syntax:

REVOKE privilege_name on object_name from {user_name | public | role_name}

Types of Database Languages: DCL & TCL

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

TCL Commands:

Commit

COMMIT is the SQL command that is used for storing changes performed by a transaction. When a COMMIT command is issued it saves all the changes since last COMMIT or ROLLBACK.

Syntax for SQL Commit

```
COMMIT;
```

RollBack

ROLLBACK is the SQL command that is used for reverting changes performed by a transaction.

When a ROLLBACK command is issued it reverts all the changes since last COMMIT or ROLLBACK.

Syntax for SQL Rollback

```
ROLLBACK;
```

SAVEPOINT

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.

Syntax:

```
SAVEPOINT SAVEPOINT_NAME;
```

Attribute Data Types in SQL

- Basic **data types**
  - **Numeric** data types
    - Integer numbers: `INTEGER`, `INT`, and `SMALLINT`
    - Floating-point (real) numbers: `FLOAT` or `REAL`, and `DOUBLE PRECISION`
  - **Character-string** data types
    - Fixed length: `CHAR(`*n*`)`, `CHARACTER(`*n*`)`
    - Varying length: `VARCHAR(`*n*`)`, `CHAR VARYING(`*n*`)`, `CHARACTER VARYING(`*n*`)`

Attribute Data Types in SQL

- **Bit-string** data types
  - Fixed length: `BIT(n)`
  - Varying length: `BIT VARYING(n)`
- **Boolean** data type
  - Values of `TRUE` or `FALSE` or `NULL`
- **DATE** data type
  - Ten positions
  - Components are `YEAR`, `MONTH`, and `DAY` in the form YYYY-MM-DD
  - Multiple mapping functions available in RDBMSs to change date formats

Attribute Data Types in SQL

PES UNIVERSITY

- Additional data types
  - **Timestamp** data type

  Includes the `DATE` and `TIME` fields
    - Plus a minimum of six positions for decimal fractions of seconds
    - Optional `WITH TIME ZONE` qualifier
  - **INTERVAL** data type
    - Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp
  - **DATE, TIME, Timestamp, INTERVAL** data types can be **cast** or converted to string formats for comparison.

PES UNIVERSITY

Large-Object Types

Many current-generation database applications need to store attributes that can be large (of the order of many kilobytes), such as a photograph, or very large (of the order of many megabytes or even gigabytes), such as a high-resolution medical image or video clip.

SQL therefore provides large-object data types for character data (**clob**) and binary data (**blob**). The letters "lob" in these data types stand for "Large OBject."

PES UNIVERSITY

Large-Object Types

For example, we may declare attributes

book review **clob**(10KB)
image **blob**(10MB)
movie **blob**(2GB)

For result tuples containing large objects (multiple megabytes to gigabytes), it is inefficient or impractical to retrieve an entire large object into memory.

.

Attribute Data Types in SQL

Advanced Data Types like CLOB, BLOB

LOBs in the database are stored in a way that optimizes the space and provides efficient access within the database tablespaces.
Internal LOBs (BLOBs, CLOBs) also provide transactional support (Commit, Rollback, and so on) of the database server.

- **BLOBs (Binary LOBs)** used to store unstructured binary (also called “raw”) data, such as video clips.
- **CLOBs (Character LOBs)** used to store large blocks of character data from the database character set.

Binary Large Object (BLOB)	Stores any kind of data in binary format such as images, audio, and video.
Character Large Object (CLOB)	Stores string data in the database having character set format. Used for large set of characters/strings or documents that use the database character.

Attribute Data Types in SQL

Advanced Data Types like CLOB, BLOB

Blob and Clob together are known as LOB(Large Object Type). The following are the major differences between Blob and Clob data types.

Blob	Clob
The full form of Blob is a Binary Large Object.	The full form of Clob is Character Large Object.
This is used to store large binary data.	This is used to store large textual data.
This stores values in the form of binary streams.	This stores values in the form of character streams.
Using this you can stores files like videos, images, gifs, and audio files.	Using this you can store files like text files, PDF documents, word documents etc.
•MySQL supports this with the following datatypes:TINYBLOB •BLOB •MEDIUMBLOB •LONGBLOB	•MySQL supports this with the following datatypes:TINYTEXT •TEXT •MEDIUMTEXT •LONGTEXT

Attribute Data Types in SQL

Advanced Data Types like CLOB, BLOB

Blob and Clob together are known as LOB(Large Object Type). The following are the major differences between Blob and Clob data types.

In JDBC API it is represented by java.sql.Blob Interface.

The Blob object in JDBC points to the location of BLOB instead of holding its binary data.

•To store Blob JDBC (PreparedStatement) provides methods like:setBlob()
•setBinaryStream()

In JDBC it is represented by java.sql.Clob Interface.

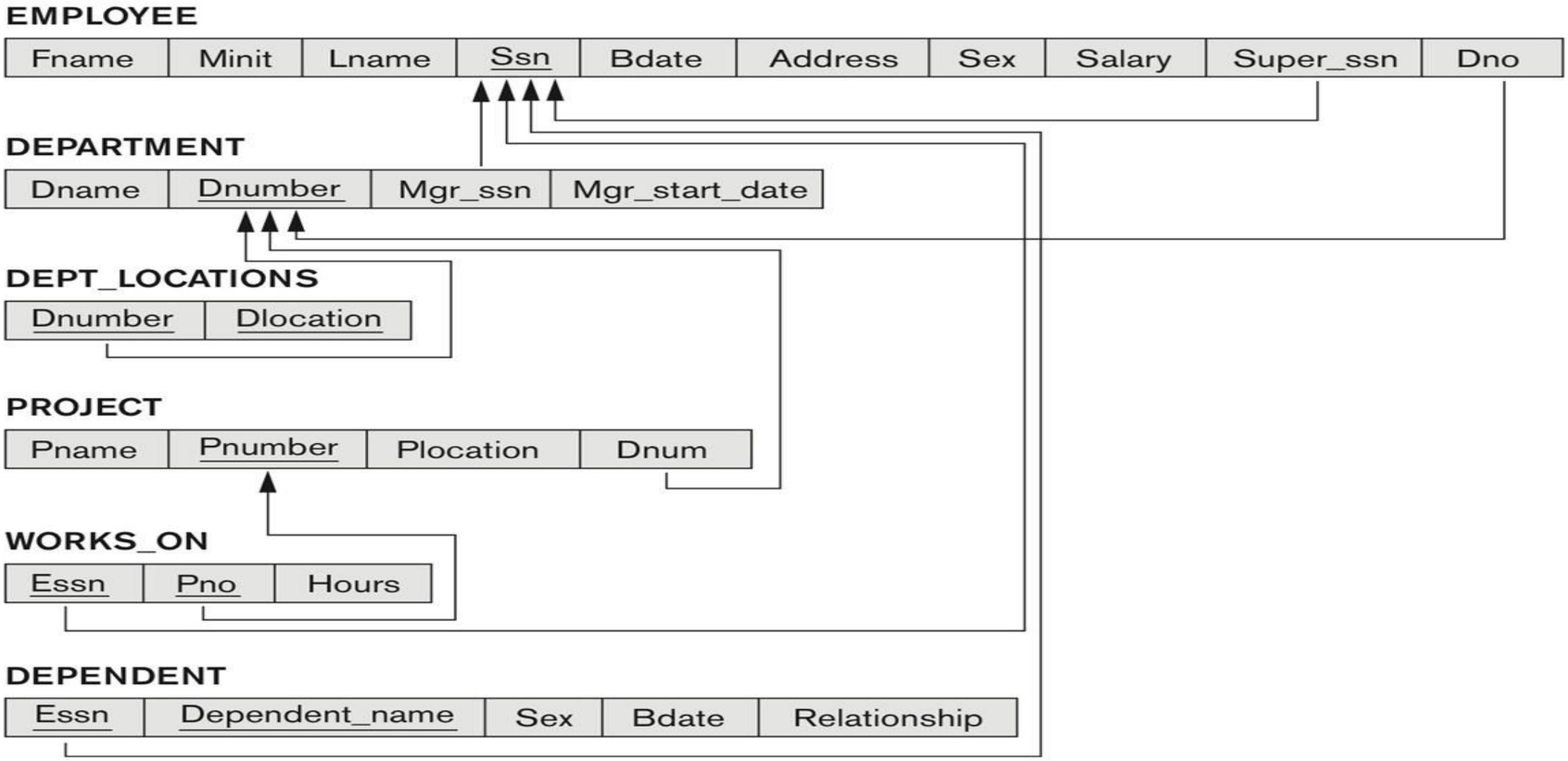
The Blob object in JDBC points to the location of BLOB instead of holding its character data.

•To store Clob JDBC (PreparedStatement) provides methods like:setClob()
•setCharacterStream()

Domains in SQL

- **Domain**
  - Name used with the attribute specification
  - Makes it easier to change the data type for a domain that is used by numerous attributes
  - Improves schema readability
  - Example:
    - `CREATE DOMAIN SSN_TYPE AS CHAR(9);`
- **TYPE**
  - User Defined Types (UDTs) are supported for object-oriented applications. Uses the command: `CREATE TYPE`

COMPANY relational database schema (Fig. 5.7)



One possible database state for the COMPANY relational database schema (Fig. 5.6)

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

One possible database state for the COMPANY relational database schema (Fig. 5.6)

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 5.7 (Fig. 6.1)

```
CREATE TABLE EMPLOYEE
        ( Fname                 VARCHAR(15)             NOT NULL,
          Minit                 CHAR,
          Lname                 VARCHAR(15)             NOT NULL,
          Ssn                   CHAR(9)                 NOT NULL,
          Bdate                 DATE,
          Address               VARCHAR(30),
          Sex                   CHAR,
          Salary                DECIMAL(10,2),
          Super_ssn             CHAR(9),
          Dno                   INT                     NOT NULL,
        PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)             NOT NULL,
          Dnumber               INT                     NOT NULL,
          Mgr_ssn               CHAR(9)                 NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber               INT                     NOT NULL,
          Dlocation             VARCHAR(15)             NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 5.7 (Fig. 6.1)

```
CREATE TABLE PROJECT
        ( Pname                         VARCHAR(15)             NOT NULL,
          Pnumber                       INT                     NOT NULL,
          Plocation                     VARCHAR(15),
          Dnum                          INT                     NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                          CHAR(9)                 NOT NULL,
          Pno                           INT                     NOT NULL,
          Hours                         DECIMAL(3,1)            NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                          CHAR(9)                 NOT NULL,
          Dependent_name                VARCHAR(15)             NOT NULL,
          Sex                           CHAR,
          Bdate                         DATE,
          Relationship                  VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

SET Operators in SQL

PES UNIVERSITY

SQL Set Operation

The SQL Set operation is used to combine the two or more SQL SELECT statements.

Types of Set Operation

1. Union
2. UnionAll
3. Intersect
4. Minus
5. Except

SET Operators in SQL

1. Union

The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
The union operation eliminates the duplicate rows from its resultset.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

SET Operators in SQL

3. Intersect

It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.

In the Intersect operation, the number of datatype and columns must be the same.

It has no duplicates and it arranges the data in ascending order by default.

Syntax

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

4. Minus

It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.

It has no duplicates and data arranged in ascending order by default.

Syntax:

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

SET Operators in SQL

5) EXCEPT

Syntax

The basic syntax of EXCEPT is as follows.

```
SELECT column1 [, column2 ] FROM table1 [, table2 ]
[WHERE condition]
EXCEPT
SELECT column1 [, column2 ] FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on user requirement.

Summary

- Introduction to SQL Commands
- SQL Data Definition, Data Types, Standards
- Schema Concepts in SQL
- The CREATE TABLE Command in SQL
- Attribute Data Types in SQL
- Advanced Data Types like CLOB, BLOB

THANK YOU

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Vinodha K & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Constraints and SQL Commands, schema changes in SQL

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr.Vinodha K & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Unit 3 : SQL

PES UNIVERSITY

1. SQL datatypes and Advanced data types like Blob Clob
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

PES UNIVERSITY

T1:6.2 -6.4,7.4

Constraints and SQL Commands, schema changes in SQL

Specifying Constraints in SQL

PES UNIVERSITY

Basic constraints:

- Relational Model has 3 basic constraint types that are supported in SQL:
  - **Key** constraint: A primary key value cannot be duplicated
  - **Entity Integrity** Constraint: A primary key value cannot be null
  - **Referential integrity** constraints : The “foreign key “ must have a value that is already present as a primary key, or may be null

Specifying Constraints in SQL

Other Restrictions on attribute domains:

- Default value of an attribute
  - **DEFAULT** <value>
  - NULL is not permitted for a particular attribute (NOT NULL)
- **CHECK** clause
  - `Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);`

Specifying Constraints in SQL

PES UNIVERSITY

- **PRIMARY KEY** clause
  - Specifies one or more attributes that make up the primary key of a relation
  - `Dnumber INT PRIMARY KEY;`
- **UNIQUE** clause
  - Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model).
  - `Dname VARCHAR(15) UNIQUE;`

Specifying Constraints in SQL

- **`FOREIGN KEY`** clause
  - Default operation: reject update on violation
  - Attach **referential triggered action** clause
    - Options include `SET NULL`, `CASCADE`, and `SET DEFAULT`
    - Action taken by the DBMS for `SET NULL` or `SET DEFAULT` is the same for both `ON DELETE` and `ON UPDATE`
    - `CASCADE` option suitable for “relationship” relations

Specifying Constraints in SQL

- Using the Keyword **CONSTRAINT**
  - Name a constraint
  - Useful for later altering

The general structure of the SQL CONSTRAINT is defined as:
The CONSTRAINT keyword is followed by a constraint name followed by a column or a list of columns.

```
CREATE TABLE EMPLOYEE
 ( … ,
 Dno INT NOT NULL DEFAULT 1,
 CONSTRAINT EMPPK  PRIMARY KEY (Ssn),
 CONSTRAINT EMPSUPERFK
 FOREIGN KEY (Super_ssn) REFERENCES
EMPLOYEE(Ssn)
 ON DELETE SET NULL ON UPDATE
CASCADE,
 CONSTRAINT EMPDEPTFK
 FOREIGN KEY(Dno) REFERENCES
DEPARTMENT(Dnumber)
 ON DELETE SET DEFAULT ON UPDATE
CASCADE);
```

Database Management Systems

Example for Specifying Constraints in SQL

Types of SQL CONSTRAINTS

The SQL provides following types of CONSTRAINTS :

Constraint	Description
NOT NULL	This constraint confirms that a column cannot store NULL value.
UNIQUE	This constraint ensures that each row for a column must have a different value.
PRIMARY KEY	This constraint is a combination of a NOT NULL constraint and a UNIQUE constraint. This constraint ensures that the specific column or combination of two or more columns for a table have a unique identity which helps to find a particular record in a table more easily and quickly.
CHECK	A check constraint ensures that the value stored in a column meets a specific condition.
DEFAULT	This constraint provides a default value when specified none for this column.
FOREIGN KEY	A foreign key constraint is used to ensure the referential integrity of the data. in one table to match values in another table.

```
CREATE TABLE mytest(
agent_code char(6) NOT NULL UNIQUE ,
agent_name char(25) NOT NULL UNIQUE ,
working_area char(25) DEFAULT 'Mumbai',
commission decimal(8,2)
CHECK(commission>.1 AND commission<.3));
```

```
CREATE TABLE mytest(
agent_code char(6) NOT NULL UNIQUE ,
agent_name char(25) NOT NULL UNIQUE ,
working_area char(25) CHECK
( working_area IN('London','Brisban','Chennai','Mumbai')) ,
commission decimal CHECK(commission<1));
```

Specifying Constraints in SQL


```
CREATE TABLE DEPT_LOCATIONS ( Dnumber INT NOT NULL,
Dlocation VARCHAR(15) NOT NULL DEFAULT ‘Banglore’,
 PRIMARY KEY (Dnumber, Dlocation),
FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

Specifying Constraints on Tuples Using CHECK

- Additional Constraints on individual tuples within a relation are also possible using CHECK
- `CHECK` clauses at the end of a `CREATE TABLE` statement
  - Apply to each tuple individually
  - `CHECK (Dept_create_date <= Mgr_start_date);`

Schema Change Statements in SQL (from chapter 7)

- **Schema evolution commands**
  - DBA may want to change the schema while the database is operational
  - Does not require recompilation of the database schema

The DROP Command

- `DROP` command
  - Used to drop named schema elements, such as tables, domains, or constraint
- Drop behavior options:
  - `CASCADE` and `RESTRICT`
- Example:
  - `DROP SCHEMA COMPANY CASCADE;`
  - This removes the schema and all its elements including tables,views, constraints, etc.

The DROP Command


```
SQL> create table gender_tab (
  2    gender_id char(1),
  3    gender_nm varchar2(6),
  4    constraint gender_pk primary key (gender_id),
  5    constraint gender_id_ck check (gender_id in ('M',
'F'))
  6  );
 Table created.
 SQL>
SQL> insert into gender_tab values ('F', 'Female');
 1 row created.
 SQL> insert into gender_tab values ('M', 'Male');
 1 row created.
 SQL>
```

The DROP Command


```
SQL> create table people (
first_name          varchar2(20),
last_name           varchar2(25),
gender          char(1)
 );

Table created.
SQL>
SQL> alter table people
   add constraint people_gender_fk
   foreign key (gender)
   references gender_tab;

Table altered.
```

Database Management Systems

The DROP Command


```
SQL> insert into people values ('S', 'Dillon', 'M');
 1 row created.
 SQL> insert into people values ('C', 'Beck', 'M');
 1 row created.
 SQL> insert into people values ('N', 'Ellis', 'F');
 1 row created.
 SQL>
SQL> drop table gender_tab;
drop table gender_tab
           *
ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys

SQL>
SQL> drop table gender_tab cascade constraints;

Table dropped.
```

MySql Commands

To create a new database in MySQL use the **CREATE DATABASE** statement with the below syntax:
CREATE DATABASE [IF NOT EXISTS] database_name

```
mysql> CREATE DATABASE employeedb;
Query OK, 1 row affected (0.26 sec)
```

To show the database created in MySQL

mysql> SHOW CREATE DATABASE employeedb;

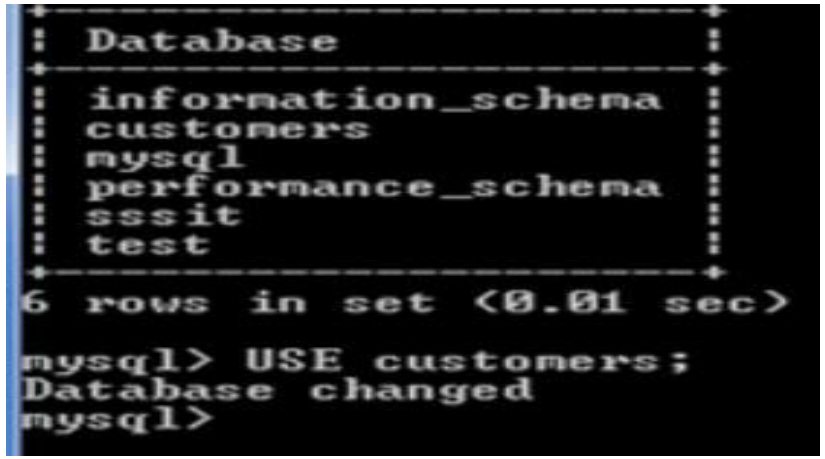
To check the created database using the following query:
mysql> SHOW DATABASES;

MySql Commands

SQL command **USE** is used to select a particular database:

Syntax:

USE database_name;



To create table in the database the following query is used:

Create table table_name(…….)

To drop the database

DROP DATABASE [IF EXISTS] database_name;

Example:

Drop database customers;

The ALTER table command

- **Alter table actions** include:
  - Adding or dropping a column (attribute)
  - Changing a column definition
  - Adding or dropping table constraints
- Example:
  - `ALTER TABLE people ADD COLUMN Job VARCHAR(12);`

The ALTER table command

1) ADD a column in the table

```
ALTER TABLE table_name ADD new_column_name column_definition
[ FIRST | AFTER column_name ];
```

2) Add multiple columns in the table

```
ALTER TABLE table_name   ADD new_column_name column_definition
[ FIRST | AFTER column_name ],
ADD new_column_name column_definition   [ FIRST | AFTER column_name ],
```

3) MODIFY column in the table

```
ALTER TABLE table_name  MODIFY column_name column_definition
[ FIRST | AFTER column_name ];
```

4) DROP column in table

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

The ALTER table command

5) RENAME column in table

ALTER TABLE table_name CHANGE COLUMN old_name new_name column_definition

[FIRST | AFTER column_name]

6) RENAME table

ALTER TABLE table_name
RENAME TO new_table_name;

7) **Altering (Changing) a Column Definition or a Name**

To change a column's definition, use **MODIFY** or **CHANGE** clause along with the ALTER command.

For example, to change column **c** from CHAR(1) to CHAR(10), you can use the following command

```
ALTER TABLE table_name MODIFY c CHAR(10);
ALTER TABLE table_name CHANGE i j BIGINT;
ALTER TABLE table_name CHANGE j j INT;
ALTER TABLE table_name MODIFY j BIGINT NOT NULL DEFAULT 100;
ALTER TABLE table_name ALTER i SET DEFAULT 1000;
```

*Note:i,j are columns

Adding and Dropping Constraints

- Change constraints specified on a table
  - Add or drop a named constraint

```
ALTER table people add constraint people_gender_fk foreign
key (gender) references gender_tab;
```

```
ALTER TABLE Persons DROP CONSTRAINT PK_Person;
```

Dropping Columns, Default Values

- To drop a column
  - Choose either `CASCADE` or `RESTRICT`
  - `CASCADE` would drop the column from views etc. `RESTRICT` is possible if no views refer to it.

ALTER TABLE EMPLOYEE **DROP COLUMN** Address **CASCADE**;

- Default values can be dropped and altered :

ALTER TABLE DEPARTMENT **ALTER COLUMN** Mgr_ssn **DROP DEFAULT**;

ALTER TABLE DEPARTMENT **ALTER COLUMN** Mgr_ssn **SET DEFAULT** '333445555';

Basic Retrieval Queries in SQL

- `SELECT` statement
  - One basic statement for retrieving information from a database
- SQL allows a table to have two or more tuples that are identical in all their attribute values
  - Relational model is strictly set-theory based
  - Multiset or bag behavior
  - Tuple-id may be used as a key

The SELECT-FROM-WHERE Structure of Basic SQL Queries

PES UNIVERSITY

- Basic form of the `SELECT` statement:

SELECT <attribute list>
FROM <table list>
WHERE <condition>;

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

The SELECT-FROM-WHERE Structure of Basic SQL Queries (cont'd.)

- Logical comparison operators
  - =, <, <=, >, >=, and <>
- **Projection attributes**
  - Attributes whose values are to be retrieved
- **Selection condition**
  - Boolean condition that must be true for any retrieved tuple. Selection conditions include join conditions (see Ch.8) when multiple relations are involved.

Basic Retrieval Queries in SQL

Bdate	Address
1965-01-09	731Fondren, Houston, TX

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:   SELECT   Bdate, Address
      FROM     EMPLOYEE
      WHERE    Fname='John' AND Minit='B' AND Lname='Smith';
```

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:   SELECT   Fname, Lname, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    Dname='Research' AND Dnumber=Dno;
```

The condition Dnumber = Dno is called a ***join condition***,
because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE,
whenever the value of Dnumber in DEPARTMENT is equal to the value of Dno in
EMPLOYEE.

Basic Retrieval Queries in SQL

(c)

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:   SELECT   Pnumber, Dnum, Lname, Address, Bdate
      FROM     PROJECT, DEPARTMENT, EMPLOYEE
      WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND
               Plocation='Stafford';
```

Ambiguous Attribute Names

- Same name can be used for two (or more) attributes in different relations
  - As long as the attributes are in different relations
  - Must **qualify** the attribute name with the relation name to prevent ambiguity

```
Q1A:  SELECT  Fname, EMPLOYEE.Name, Address
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DEPARTMENT.Name='Research' AND
              DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Aliasing, and Renaming

- **Aliases** or **tuple variables**
  - Declare alternative relation names E and S to refer to the EMPLOYEE relation twice in a query:

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT  E.Fname, E.Lname, S.Fname, S.Lname
FROM    EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn=S.Ssn;
```

  - Recommended practice to abbreviate names and to prefix same or similar attribute from multiple tables.

Aliasing, Renaming and Tuple Variables (contd.)

- The attribute names can also be renamed

```
EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn,
  Dno)
```

- Note that the relation EMPLOYEE now has a variable name E which corresponds to a tuple variable
- The “AS” may be dropped in most SQL implementations

Unspecified WHERE Clause and Use of the Asterisk

- Missing `WHERE` clause
  - Indicates no condition on tuple selection
- Effect is a `CROSS PRODUCT`
  - Result is all possible tuple combinations

Queries 9 and 10. Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

```
Q9:     SELECT    Ssn
        FROM      EMPLOYEE;

Q10:    SELECT    Ssn, Dname
        FROM      EMPLOYEE, DEPARTMENT;
```

Unspecified WHERE Clause and Use of the Asterisk

- Specify an asterisk (*)
  - Retrieve all the attribute values of the selected tuples
  - The * can be prefixed by the relation name; e.g., EMPLOYEE *

```
Q1C:    SELECT    *
        FROM      EMPLOYEE
        WHERE     Dno=5;

Q1D:    SELECT    *
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     Dname='Research' AND Dno=Dnumber;

Q10A:   SELECT    *
        FROM      EMPLOYEE, DEPARTMENT;
```

Tables as Sets in SQL

PES UNIVERSITY

- SQL does not automatically eliminate duplicate tuples in query results
- For aggregate operations duplicates will be accounted in end results.
- Use the keyword **`DISTINCT`** in the `SELECT` clause
  - Only distinct tuples should remain in the result

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

```
Q11:   SELECT    ALL Salary
       FROM      EMPLOYEE;

Q11A:  SELECT    DISTINCT Salary
       FROM      EMPLOYEE;
```

Tables as Sets in SQL

- Set operations
  - **UNION**, **EXCEPT** (difference), **INTERSECT**
  - Corresponding multiset operations: `UNION ALL`, `EXCEPT ALL`, `INTERSECT ALL`)
  - Type compatibility is needed for these operations to be valid

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  (SELECT    DISTINCT Pnumber
       FROM      PROJECT, DEPARTMENT, EMPLOYEE
       WHERE     Dnum=Dnumber AND Mgr_ssn=Ssn
                 AND Lname='Smith' )
       UNION
      ( SELECT   DISTINCT Pnumber
       FROM      PROJECT, WORKS_ON, EMPLOYEE
       WHERE     Pnumber=Pno AND Essn=Ssn
                 AND Lname='Smith' );
```

Substring Pattern Matching and Arithmetic Operators

- **`LIKE`** comparison operator
  - Used for string **pattern matching**
  - % replaces an arbitrary number of zero or more characters
  - underscore (_) replaces a single character
  - Examples: **WHERE** Address **LIKE** '%Houston,TX%';
  - **WHERE** Ssn **LIKE** '_ _ 1_ _ 8901';
- **`BETWEEN`** comparison operator

E.g., in Q14 :

WHERE(Salary **BETWEEN** 30000 **AND** 40000)
AND Dno = 5;

Arithmetic Operations

- Standard arithmetic operators:
  - Addition (+), subtraction (–), multiplication (*), and division (/) may be included as a part of **SELECT**

- **Query 13.** Show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10 percent raise.

```
SELECT  E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM  EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE  E.Ssn=W.Essn AND W.Pno=P.Pnumber AND P.Pname=‘ProductX’;
```

Basic SQL Retrieval Query Block

```
SELECT      <attribute list>
FROM        <table list>
[ WHERE     <condition> ]
[ ORDER BY <attribute list> ];
```

Ordering of Query Results

- Use **ORDER BY** clause
  - Keyword **DESC** to see result in a descending order of values
  - Keyword **ASC** to specify ascending order explicitly
  - Typically placed at the end of the query

```
ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC
```

INSERT, DELETE, and UPDATE Statements in SQL

- Three commands used to modify the database:
  - `INSERT`, `DELETE`, and `UPDATE`
- `INSERT` typically inserts a tuple (row) in a relation (table)
- `UPDATE` may update a number of tuples (rows) in a relation (table) that satisfy the condition
- `DELETE` may also update a number of tuples (rows) in a relation (table) that satisfy the condition

INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the **CREATE TABLE** command
- Constraints on data types are observed automatically
- Any integrity constraints as a part of the DDL specification are enforced

The INSERT Command

- Specify the relation name and a list of values for the tuple. All values including nulls are supplied.

```
U1:   INSERT INTO   EMPLOYEE
      VALUES        ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                      Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

- The variation below inserts multiple tuples where a new table is loaded values from the result of a query.

```
U3B:  INSERT INTO   WORKS_ON_INFO ( Emp_name, Proj_name,
                    Hours_per_week )
      SELECT        E.Lname, P.Pname, W.Hours
      FROM          PROJECT P, WORKS_ON W, EMPLOYEE E
      WHERE         P.Pnumber=W.Pno AND W.Essn=E.Ssn;
```

The Insert Command


```
CREATE TABLE WORKS_ON_INFO
( Emp_name VARCHAR(15),
Proj_name VARCHAR(15),
Hours_per_week DECIMAL(3,1) );
```

```
INSERT INTO WORKS_ON_INFO ( Emp_name, Proj_name, Hours_per_week )

SELECT E.Lname, P.Pname, W.Hours FROM PROJECT P, WORKS_ON W,
EMPLOYEE E

WHERE P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

BULK LOADING OF TABLES

- Another variation of **INSERT** is used for bulk-loading of several tuples into tables
- A new table TNEW can be created with the same attributes as T and using LIKE and DATA in the syntax, it can be loaded with entire data.
- EXAMPLE:

```
CREATE TABLE D5EMPS  LIKE  EMPLOYEE
        (SELECT   E.*
         FROM      EMPLOYEE AS E
        WHERE     E.Dno=5)
 WITH DATA;
```

```
company=# CREATE TABLE D5EMPS as
company-# (SELECT E.*
company(# FROM EMPLOYEE AS E
company(# WHERE E.Dno = 5) WITH DATA;
SELECT 1
company=# select * from D5EMPS;
 fname | mname | lname |  ssn   | address  |  salary  | super_ssn | dno
-------+-------+-------+--------+----------+----------+-----------+-----
 joe   | M     | rao   | 334455 | banglore | 32980.09 | 334455    |   5
(1 row)
```

DELETE

- Removes tuples from a relation
  - Includes a WHERE-clause to select the tuples to be deleted
  - Referential integrity should be enforced
  - Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
  - A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
  - The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

The DELETE Command

- Removes tuples from a relation
  - Includes a `WHERE` clause to select the tuples to be deleted. The number of tuples deleted will vary.

```
U4A:  DELETE FROM  EMPLOYEE
      WHERE        Lname='Brown';

U4B:  DELETE FROM  EMPLOYEE
      WHERE        Ssn='123456789';

U4C:  DELETE FROM  EMPLOYEE
      WHERE        Dno=5;

U4D:  DELETE FROM  EMPLOYEE;
```

UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity specified as part of DDL specification is enforced

UPDATE (contd.)

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively

```
U5:     UPDATE    PROJECT
        SET       PLOCATION = 'Bellaire',
    DNUM = 5
        WHERE     PNUMBER=10
```

UPDATE (contd.)

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6: UPDATE      EMPLOYEE
    SET         SALARY = SALARY *1.1
    WHERE       DNO  IN (SELECT     DNUMBER
                    FROM        DEPARTMENT
                    WHERE       DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
  - The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
  - The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

Summary

- SQL
  - A Comprehensive language for relational database management
  - Data definition, queries, updates, constraint specification, and view definition
- Covered :
  - Data definition commands for creating tables
  - Commands for constraint specification
  - Simple retrieval queries
  - Database update commands
- Schema Modification for the DBAs using **`ALTER TABLE , ADD and DROP COLUMN, ALTER CONSTRAINT`** etc.

THANK YOU

Dr.Vinodha K & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Vinodha, Dr. Geetha, Prof.Shanthala

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Advanced SQL Queries, Nested Queries

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

DATABASE MANAGEMENT SYSTEM

Unit 3 : Advanced SQL Queries, Nested Queries

PES UNIVERSITY

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

PES UNIVERSITY

T1:7.1.1,7.1.2

Advanced SQL Queries, Nested Queries

Database Management Systems

More Complex SQL Retrieval Queries

PES UNIVERSITY

- Additional features allow users to specify more complex retrievals from database:
  - Nested queries, joined tables, and outer joins (in the FROM clause), aggregate functions, and grouping

Database Management Systems

Comparisons Involving NULL and Three-Valued Logic

- Meanings of NULL
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN (like Maybe)
- **NULL = NULL comparison is avoided**

Database Management Systems

Comparisons Involving NULL and Three-Valued Logic (cont'd.)

Table 7.1 Logical Connectives in Three-Valued Logic

(a)	**AND**	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	FALSE	UNKNOWN
	FALSE	FALSE	FALSE	FALSE
	UNKNOWN	UNKNOWN	FALSE	UNKNOWN
(b)	**OR**	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	UNKNOWN
	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
(c)	**NOT**			
	TRUE	FALSE		
	FALSE	TRUE		
	UNKNOWN	UNKNOWN		

Database Management Systems

Comparisons Involving NULL and Three-Valued Logic (cont'd.)

- SQL allows queries that check whether an attribute value is NULL
  - IS or IS NOT NULL

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

Database Management Systems

Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries**
  - complete select-from-where blocks within another SQL query.
  - That other query is called the outer query.
  - These nested queries can also appear in the WHERE clause or the FROM clause or the SELECT clause or other SQL clauses as needed.
  - Comparison operator IN
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to TRUE if *v* is one of the elements in *V*

Database Management Systems

Nested Queries (cont'd.)

PES UNIVERSITY

```
Q4A:  SELECT   DISTINCT Pnumber
      FROM     PROJECT
      WHERE    Pnumber IN
               ( SELECT     Pnumber
                 FROM       PROJECT, DEPARTMENT, EMPLOYEE
                 WHERE      Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
               OR
               Pnumber IN
               ( SELECT     Pno
                 FROM       WORKS_ON, EMPLOYEE
                 WHERE      Essn=Ssn AND Lname='Smith' );
```

Database Management Systems

Nested Queries (cont'd.)

Use tuples of values in comparisons

- Place them within parentheses

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```

Database Management Systems

Nested Queries (cont'd.)

- Use other comparison operators to compare a single value *v*
  - = ANY (or = SOME) operator
    - Returns TRUE if the value *v* is equal to some value in the set *V* and is hence equivalent to IN
  - Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>
  - ALL: value must exceed all values from nested query

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL  ( SELECT    Salary
                          FROM      EMPLOYEE
                          WHERE     Dno=5 );
```

Database Management Systems

Nested Queries (cont'd.)

- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:  SELECT  E.Fname, E.Lname
      FROM    EMPLOYEE AS E
      WHERE   E.Ssn IN  ( SELECT  Essn
                          FROM    DEPENDENT AS D
                          WHERE   E.Fname=D.Dependent_name
                                  AND E.Sex=D.Sex );
```

THANK YOU

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Geetha & Dr.Vinodha
Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

SQL : Sub- queries, Correlated sub-queries

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

DATABASE MANAGEMENT SYSTEM

Unit 3 : Advanced SQL Queries, Nested Queries

PES UNIVERSITY

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

PES UNIVERSITY

T1 :7.1.3 -7.1.5

Sub- queries, Correlated sub-queries

Correlated Nested Queries

- **Correlated** nested query

Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated.

- Evaluated once for each tuple in the outer query

Database Management Systems

Correlated Nested Queries

- **Queries that are nested using the = or IN comparison operator** can be collapsed into one single block.
- E.g., For each employee tuple, evaluate the nested query, which retrieves the Essn values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple; if the Ssn value of the
- EMPLOYEE tuple is in the result of the nested query, then select that EMPLOYEE tuple.

```
Q16A:   SELECT   E.Fname, E.Lname
        FROM     EMPLOYEE AS E, DEPENDENT AS D
        WHERE    E.Ssn=D.Essn AND E.Sex=D.Sex
                 AND    E.Fname=D.Dependent_name;
```

Database Management Systems

The EXISTS and UNIQUE Functions in SQL for correlating queries

PES UNIVERSITY

- EXISTS function
- EXISTS and UNIQUE are Boolean functions that return TRUE or FALSE; hence, they can be used in a WHERE clause condition

- Check whether the result of a correlated nested query is empty or not. They are Boolean functions that return a TRUE or FALSE result.

- The result of EXISTS is a Boolean value TRUE if the nested query result contains at least one tuple, or FALSE if the nested query result contains no tuples

Database Management Systems

The EXISTS and UNIQUE Functions in SQL for correlating queries

- EXISTS and NOT EXISTS
  - Typically used in conjunction with a correlated nested query
- SQL function UNIQUE(Q)
  - Returns TRUE if there are no duplicate tuples in the result of query Q

```
Q16B:   SELECT   E.Fname, E.Lname
        FROM     EMPLOYEE AS E
        WHERE    EXISTS ( SELECT   *
                          FROM     DEPENDENT AS D
                          WHERE    E.Ssn = D.Essn AND E.Sex = D.Sex
                          AND E.Fname = D.Dependent_name);
```

Database Management Systems

The EXISTS and UNIQUE Functions in SQL for correlating queries

Query 6. Retrieve the names of employees who have no dependents.

```
Q6:   SELECT   Fname, Lname
      FROM     EMPLOYEE
      WHERE    NOT EXISTS ( SELECT   *
                            FROM     DEPENDENT
                            WHERE    Ssn = Essn );
```

USE of EXISTS

Query 7. List the names of managers who have at least one dependent.

```
Q7:   SELECT   Fname, Lname
      FROM     EMPLOYEE
      WHERE    EXISTS ( SELECT   *
                        FROM     DEPENDENT
                        WHERE    Ssn = Essn )
               AND
               EXISTS ( SELECT   *
                        FROM     DEPARTMENT
                        WHERE    Ssn = Mgr_ssn );
```

Database Management Systems

USE OF NOT EXISTS

To achieve the "for all" (universal quantifier- see Ch.8) effect, we use double negation this way in SQL:

The query Q3: Retrieve the name of each employee who works on all the projects controlled by department number 5 can be written using EXISTS and NOT EXISTS in SQL systems

```
Q3A:  SELECT   Fname, Lname
      FROM     EMPLOYEE
      WHERE    NOT EXISTS ( ( SELECT   Pnumber
                              FROM     PROJECT
                              WHERE    Dnum = 5)
                              EXCEPT   ( SELECT   Pno
                                         FROM     WORKS_ON
                                         WHERE    Ssn = Essn) );
```

The above is equivalent to double negation: List names of those employees for whom there does NOT exist a project managed by department no. 5 that they do NOT work on.

Database Management Systems

Double Negation to accomplish "for all" in SQL

```
Q3B:  SELECT   Lname, Fname
      FROM     EMPLOYEE
      WHERE    NOT EXISTS  ( SELECT   *
                             FROM     WORKS_ON B
                             WHERE    ( B.Pno IN  ( SELECT   Pnumber
                                                    FROM     PROJECT
                                                    WHERE    Dnum = 5 )
                             AND
                             NOT EXISTS  ( SELECT   *
                                           FROM     WORKS_ON C
                                           WHERE    C.Essn = Ssn
                                           AND      C.Pno = B.Pno )));
```

In Q3B, the outer nested query selects any WORKS_ON (B) tuples whose Pno is of a project controlled by department 5, if there is not a WORKS_ON (C) tuple with the same Pno and the same Ssn as that of the EMPLOYEE tuple under consideration in the outer query. If no such tuple exists, we select the EMPLOYEE tuple.

Database Management Systems

Explicit Sets and Renaming of Attributes in SQL

- Can use explicit set of values in WHERE clause

```
Q17:    SELECT   DISTINCT Essn
        FROM     WORKS_ON
        WHERE    Pno IN (1, 2, 3);
```

- Use qualifier AS followed by desired new name
  - Rename any attribute that appears in the result of a query

Aliases for Attributes

```
Q8A:    SELECT   E.Lname AS Employee_name, S.Lname AS Supervisor_name
        FROM     EMPLOYEE AS E, EMPLOYEE AS S
        WHERE    E.Super_ssn=S.Ssn;
```

THANK YOU

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Geetha & Dr.Vinodha, Prof. Shanthala

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Outer join, Aggregation function and group, having clause

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

T1:7.1.6 - 7.1.8

Outer join, Aggregation function and group, having clause

DATABASE MANAGEMENT SYSTEM

Unit 3 : Outer join, Aggregation function and group, having clause

PES UNIVERSITY

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

Database Management Systems

Specifying Joined Tables in the FROM Clause of SQL

- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query
- The FROM clause in Q1A
  - Contains a single joined table. JOIN may also be called INNER JOIN

```
Q1A:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
        WHERE     Dname='Research';
```

Database Management Systems

Different Types of JOINed Tables in SQL

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN (LEFT, RIGHT, FULL)
- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S

Database Management Systems

NATURAL JOIN

- Rename attributes of one relation so it can be joined with another using NATURAL JOIN:

```
Q1B:    SELECT    Fname, Lname, Address
        FROM (EMPLOYEE NATURAL JOIN
                  (DEPARTMENT AS DEPT (Dname, Dno, Mssn,
                  Msdate)))
        WHERE     Dname='Research';
```

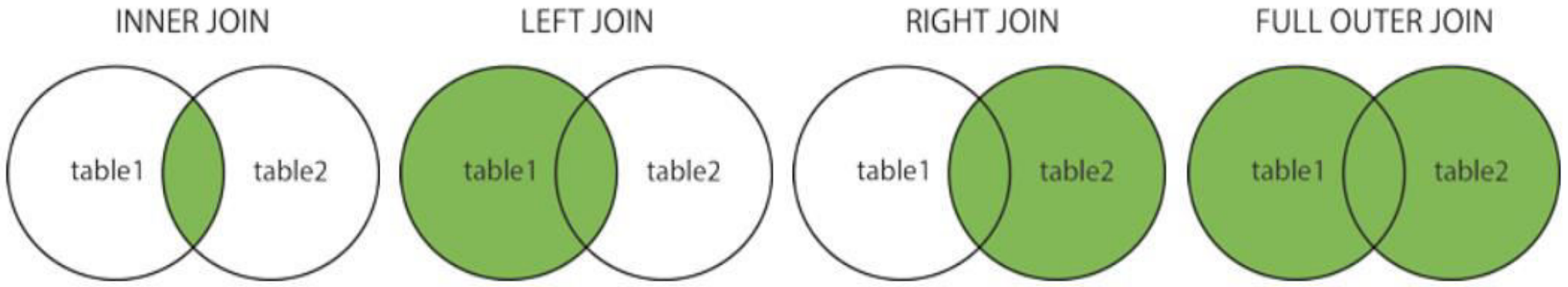
The above works with EMPLOYEE.Dno = DEPT.Dno as an implicit join condition

Database Management Systems

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(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



Database Management Systems

INNER and OUTER Joins

- INNER JOIN **(versus** OUTER JOIN**)**
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation
- LEFT OUTER JOIN
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table
- RIGHT OUTER JOIN
  - Every tuple in right table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of left table

Database Management Systems

Example: LEFT OUTER JOIN


```
SELECT E.Lname AS Employee_Name,
                S.Lname AS Supervisor_Name
FROM (Employee AS E LEFT OUTER JOIN EMPLOYEE AS S
                        ON E.Super_ssn = S.Ssn)
```

ALTERNATE SYNTAX:

```
SELECT E.Lname , S.Lname
FROM  EMPLOYEE E, EMPLOYEE S
WHERE E.Super_ssn + = S.Ssn
```

Database Management Systems

Multiway JOIN in the FROM clause

- FULL OUTER JOIN – combines result of LEFT and RIGHT OUTER JOIN
- Can nest JOIN specifications for a multiway join:

Q2A: **SELECT** Pnumber, Dnum, Lname, Address, Bdate
FROM ((PROJECT **JOIN** DEPARTMENT **ON** Dnum=Dnumber) **JOIN** EMPLOYEE **ON** Mgr_ssn=Ssn)
WHERE Plocation='Stafford';

Database Management Systems

Aggregate Functions in SQL

PES UNIVERSITY

- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
  - **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- **Grouping**
  - Create subgroups of tuples before summarizing
- To select entire groups, HAVING clause is used
- Aggregate functions can be used in the SELECT clause or in a HAVING clause

Database Management Systems

Renaming Results of Aggregation

- Following query returns a single row of computed values from EMPLOYEE table:

Q19: **SELECT** **SUM** (Salary), **MAX** (Salary), **MIN** (Salary), **AVG** (Salary)
FROM EMPLOYEE;

- The result can be presented with new names:

Q19A: **SELECT** **SUM** (Salary) **AS** Total_Sal, **MAX** (Salary) **AS** Highest_Sal, **MIN** (Salary) **AS** Lowest_Sal, **AVG** Salary) **AS** Average_Sal
FROM EMPLOYEE;

Database Management Systems

Aggregate Functions in SQL (cont'd.)

- NULL values are discarded when aggregate functions are applied to a particular column

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:  SELECT  SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM    (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE   Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:  SELECT  COUNT (*)
      FROM    EMPLOYEE;

Q22:  SELECT  COUNT (*)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNO=DNUMBER AND DNAME='Research';
```

Database Management Systems

Aggregate Functions on Booleans

- SOME and ALL may be applied as functions on Boolean Values.
- SOME returns true if at least one element in the collection is TRUE (similar to OR)
- ALL returns true if all of the elements in the collection are TRUE (similar to AND)

Database Management Systems

Grouping: The GROUP BY Clause

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **GROUP BY** clause
  - Specifies grouping attributes
- COUNT (*) counts the number of rows in the group

Database Management Systems

Examples of GROUP BY

- The grouping attribute must appear in the SELECT clause:

Q24: **SELECT** Dno, **COUNT** (*), **AVG** (Salary)
FROM EMPLOYEE
GROUP BY Dno;

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

Database Management Systems

Examples of GROUP BY

- If the grouping attribute has NULL as a possible value, then a separate group is created for the null value (e.g., null Dno in the above query)
- GROUP BY may be applied to the result of a JOIN:
- Query 25. For each project, retrieve the project number, the project name, and
  the number of employees who work on that project.

```
Q25:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE             Pnumber=Pno
        GROUP BY    Pnumber, Pname;
```

Q25 shows how we can use a join condition in conjunction with GROUP BY. In this case, the grouping and functions are applied *after* the joining of the two relations in the WHERE clause.

Database Management Systems

Grouping: The GROUP BY and HAVING Clauses (cont'd.)

- **HAVING** clause
  - Provides a condition to select or reject an entire group:
- **Query 26.** For each project *on which more than two employees work,* retrieve the project number, the project name, and the number of employees who work on the project.

Q26:

```
SELECT  Pnumber, Pname, COUNT (*)
FROM    PROJECT, WORKS_ON
WHERE   Pnumber=Pno
GROUP BY  Pnumber, Pname
HAVING  COUNT (*) > 2;
```

Database Management Systems

Grouping: The GROUP BY and HAVING Clauses (cont'd.)

(b)

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING

Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the HAVING clause condition

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

Database Management Systems

Combining the WHERE and the HAVING Clause

- Consider the query: we want to count the *total* number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.

- INCORRECT QUERY:

```
SELECT      Dno, COUNT (*)
FROM        EMPLOYEE
WHERE       Salary>40000
GROUP BY    Dno
HAVING      COUNT (*) > 5;
```

Database Management Systems

Combining the WHERE and the HAVING Clause (continued)

PES UNIVERSITY

Correct Specification of the Query:

- Note: the WHERE clause applies tuple by tuple whereas HAVING applies to entire group of tuples

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
Q28:  SELECT   Dnumber, COUNT (*)
      FROM     DEPARTMENT, EMPLOYEE
      WHERE    Dnumber=Dno AND Salary>40000 AND
               ( SELECT       Dno
                 FROM         EMPLOYEE
                 GROUP BY Dno
                 HAVING       COUNT (*) > 5)
```

THANK YOU

Dr.Geetha & Dr.Vinodha
Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Other SQL Constructs: WITH, CASE

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Unit 3 : Advanced SQL Queries, Nested Queries

PES UNIVERSITY

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

PES UNIVERSITY

T1:7.1.9,7.1.10

Other SQL Constructs: WITH, CASE, Recursive Queries in SQL

Database Management Systems

USE OF WITH

- The WITH clause allows a user to define a table that will only be used in a particular query (not available in all SQL implementations)
- Used for convenience to create a temporary “View” and use that immediately in a query
- Allows a more straightforward way of looking a step-by-step query

Database Management Systems

Example of WITH

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

The alternate approach for the above Q28:

```
Q28':   WITH    BIGDEPTS (Dno) AS
                (   SELECT  Dno
                    FROM    EMPLOYEE
                    GROUP BY Dno
                    HAVING  COUNT (*) > 5)
        SELECT          Dno, COUNT (*)
        FROM            EMPLOYEE
        WHERE           Salary>40000 AND Dno IN
BIGDEPTS
                GROUP BY Dno;
```

Use of CASE

- SQL also has a CASE construct
- Used when a value can be different based on certain conditions.
- Can be used in any part of an SQL query where a value is expected
- Applicable when querying, inserting or updating tuples

EXAMPLE of use of CASE

The following example shows that employees are receiving different raises in different departments (A variation of the update U6)

```
U6':    UPDATE  EMPLOYEE
        SET     Salary =
        CASE    WHEN    Dno = 5     THEN
Salary + 2000
                WHEN    Dno = 4     THEN
Salary + 1500
                WHEN    Dno = 1     THEN
Salary + 3000
```

Recursive Queries in SQL

- An example of a **recursive relationship** between tuples of the same type is the relationship between an employee and a supervisor.
- This relationship is described by the foreign key Super_ssn of the EMPLOYEE relation
- An example of a **recursive operation** is to retrieve all supervisees of a supervisory employee *e* at all levels—that is, all employees *e′* directly supervised by *e*, all employees *e′’* directly supervised by each employee *e′*, all employees *e′′′* directly supervised by each employee *e′′*, and so on.
- Thus the CEO would have each employee in the company as a supervisee in the resulting table. Example shows such table SUP_EMP with 2 columns (Supervisor,Supervisee(any level)):

Database Management Systems

An EXAMPLE of RECURSIVE Query

- **Q29:**

```
WITH RECURSIVE SUP_EMP (SupSsn, EmpSsn) AS
        SELECT      SupervisorSsn, Ssn
        FROM        EMPLOYEE
                    UNION
        SELECT      E.Ssn, S.SupSsn
        FROM        EMPLOYEE AS E, SUP_EMP AS S
        WHERE       E.SupervisorSsn = S.EmpSsn)
        SELECT      *
        FROM        SUP_EMP;
```

- The above query starts with an empty SUP_EMP and successively builds SUP_EMP table by computing immediate supervisees first, then second level supervisees, etc. until a **fixed point** is reached and no more supervisees can be added

The above query starts with an empty SUP_EMP and successively builds SUP_EMP table by computing immediate supervisees first, then second level supervisees, etc. until a **fixed point** is reached and no more supervisees can be added

THANK YOU

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Vinodha ,Dr.Geetha. Prof.Shanthala

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

CUBE, PIVOT, ROLLUP

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

1. Unit 3 : Other SQL Constructs: WITH, CASE

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

R1 : CHAPTER 5.6

CUBE, PIVOT, ROLLUP

CUBE, PIVOT, ROLLUP

PES UNIVERSITY

Online Analytical Processing (OLAP)

Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay)

Data that can be modeled as dimension attributes and measure attributes are called **multidimensional data**.

Measure attributes

measure some value

can be aggregated upon

e.g. the attribute *number* of the *sales* relation

Dimension attributes

define the dimensions on which measure attributes (or aggregates thereof) are viewed

e.g. the attributes *item_name, color,* and *size* of the *sales* relation

DATABASE MANAGEMENT SYSTEM

CUBE, PIVOT, ROLLUP

Consider an application where a shop wants to find out what kinds of clothes are popular. Let us suppose that clothes are characterized by their item name, color, and size, and that we have a relation *sales* with the schema.

sales (*item_name*, *color*, *clothes_size*, *quantity*)

item_name	*color*	*clothes_size*	*quantity*
skirt	dark	small	2
skirt	dark	medium	5
skirt	dark	large	1
skirt	pastel	small	11
skirt	pastel	medium	9
skirt	pastel	large	15
skirt	white	small	2
skirt	white	medium	5
skirt	white	large	3
dress	dark	small	2
dress	dark	medium	6
dress	dark	large	12
dress	pastel	small	4
dress	pastel	medium	3
dress	pastel	large	3
dress	white	small	2
dress	white	medium	3
dress	white	large	0
shirt	dark	small	2
shirt	dark	medium	6
shirt	dark	large	6
shirt	pastel	small	4
shirt	pastel	medium	1
shirt	pastel	large	2
shirt	white	small	17
shirt	white	medium	1
shirt	white	large	10
pants	dark	small	14
pants	dark	medium	6
pants	dark	large	0
pants	pastel	small	1
pants	pastel	medium	0
pants	pastel	large	1
pants	white	small	3
pants	white	medium	0
pants	white	large	2

Figure 5.16 An example of *sales* relation.

CUBE, PIVOT, ROLLUP

Cross Tabulation of *sales* by *item-name* and *color*

size: **all**

item-name \ *color*	dark	pastel	white	Total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pant	20	2	5	27
Total	62	54	48	164

The table above is an example of a **cross-tabulation** (**cross-tab**), also referred to as a **pivot-table**.

Values for one of the dimension attributes form the row headers

Values for another dimension attribute form the column headers

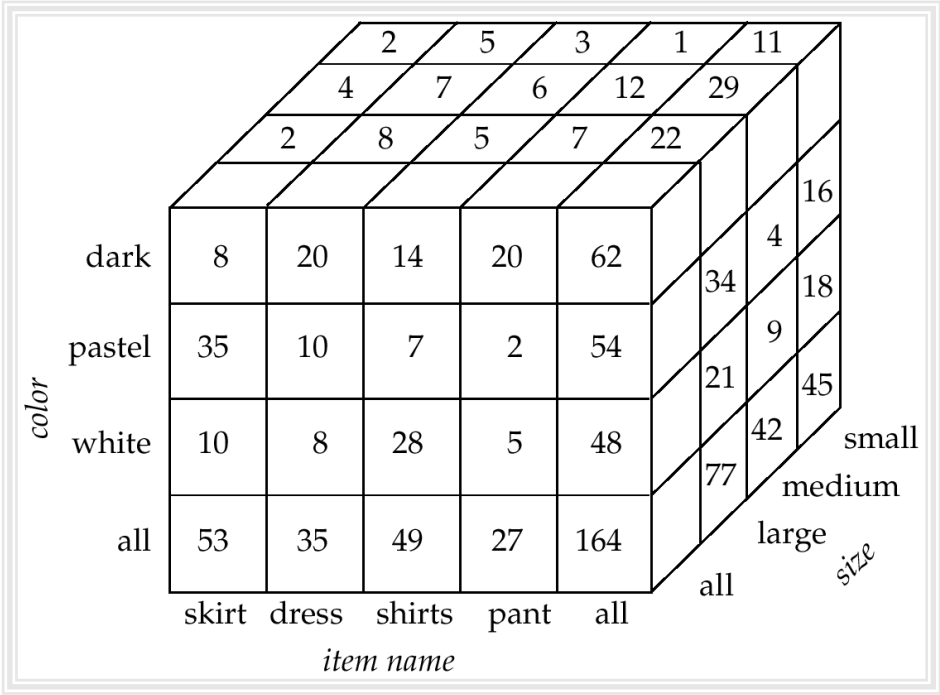
Other dimension attributes are listed on top

Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.

CUBE, PIVOT, ROLLUP

Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab
- Can have *n* dimensions; we show 3 below
- Cross-tabs can be used as views on a data cube



CUBE, PIVOT, ROLLUP

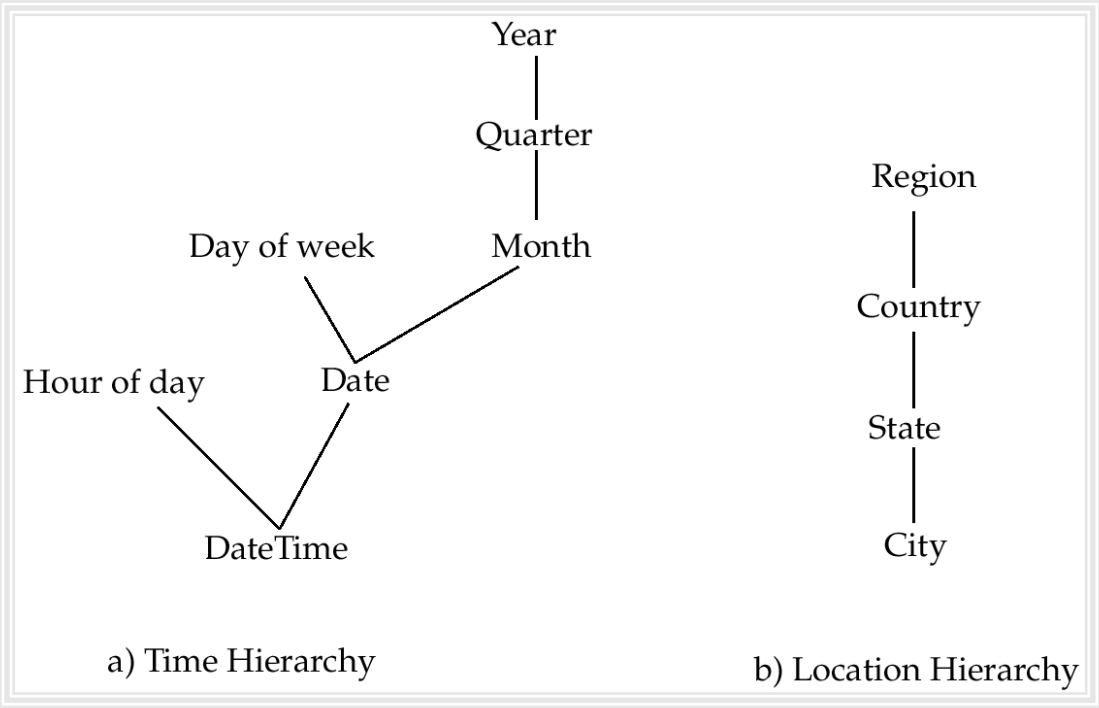
PES UNIVERSITY

Online Analytical Processing

- **Pivoting**: changing the dimensions used in a cross-tab is called
- **Slicing:** creating a cross-tab for fixed values only
  - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- **Rollup**: moving from finer-granularity data to a coarser granularity
- **Drill down**: The opposite operation - that of moving from coarser-granularity data to finer-granularity data

CUBE, PIVOT, ROLLUP Hierarchies on Dimensions

- **Hierarchy** on dimension attributes: lets dimensions to be viewed at different levels of detail
  - E.g. the dimension DateTime can be used to aggregate by hour of day, date, day of week, month, quarter or year



CUBE, PIVOT, ROLLUP

Cross Tabulation With Hierarchy

- Cross-tabs can be easily extended to deal with hierarchies
  - Can drill down or roll up on a hierarchy

category	*item-name*					
		dark	pastel	white	total	
womenswear	skirt	8	8	10	53	
	dress	20	20	5	35	
	subtotal	28	28	15		88
menswear	pants	14	14	28	49	
	shirt	20	20	5	27	
	subtotal	34	34	33		76
total		62	62	48		164

CUBE, PIVOT, ROLLUP

Relational Representation of Cross-tabs

- Cross-tabs can be represented as relations
  - We use the value **all** is used to represent aggregates
  - The SQL:1999 standard actually uses null values in place of **all** despite confusion with regular null values

item-name	*color*	*number*
skirt	dark	8
skirt	pastel	35
skirt	white	10
skirt	**all**	53
dress	dark	20
dress	pastel	10
dress	white	5
dress	**all**	35
shirt	dark	14
shirt	pastel	7
shirt	white	28
shirt	**all**	49
pant	dark	20
pant	pastel	2
pant	white	5
pant	**all**	27
all	dark	62
all	pastel	54
all	white	48
all	**all**	164

PES UNIVERSITY

OLAP Implementation

- The earliest OLAP systems used multidimensional arrays in memory to store data cubes, and are referred to as multidimensional OLAP (MOLAP) systems.
- OLAP implementations using only relational database features are called relational OLAP (ROLAP) systems
- Hybrid systems, which store some summaries in memory and store the base data and other summaries in a relational database, are called hybrid OLAP (HOLAP) systems.

CUBE, PIVOT, ROLLUP

OLAP Implementation (Cont.)

- Early OLAP systems precomputed *all* possible aggregates in order to provide online response
  - Space and time requirements for doing so can be very high
    - 2^n combinations of **group by**
  - It suffices to precompute some aggregates, and compute others on demand from one of the precomputed aggregates
    - Can compute aggregate on (*item-name, color*) from an aggregate on (*item-name, color, size*)
      - For all but a few "non-decomposable" aggregates such as *median*
      - is cheaper than computing it from scratch
- Several optimizations available for computing multiple aggregates
  - Can compute aggregate on (*item-name, color*) from an aggregate on (*item-name, color, size*)
  - Can compute aggregates on (*item-name, color, size*), (*item-name, color*) and (*item-name*) using a single sorting of the base data

CUBE, PIVOT, ROLLUP

OLAP Implementation (Cont.)

OLAP in SQL

Given the *sales* relation, the query

```
select *
from sales
pivot (
        sum(quantity)
        for color in ('dark','pastel','white')
)
order by item_name;
```

item_name	clothes_size	dark	pastel	white
skirt	small	2	11	2
skirt	medium	5	9	5
skirt	large	1	15	3
dress	small	2	4	2
dress	medium	6	3	3
dress	large	12	3	0
shirt	small	2	4	17
shirt	medium	6	1	1
shirt	large	6	2	10
pants	small	14	1	3
pants	medium	6	0	0
pants	large	0	1	2

Result of SQL pivot operation on the *sales* relation

CUBE, PIVOT, ROLLUP

The data in a data cube cannot be generated by a single SQL query, using the basic **group by** constructs, since aggregates are computed for several different groupings of the dimension attributes.

For this reason, SQL includes functions to form the grouping needed for OLAP.

SQL supports generalizations of the **group by** construct to perform the **cube** and **rollup** operations. The **cube** and **rollup** constructs in the **group by** clause

allow multiple **group by** queries to be run in a single query with the result returned as a single relation

CUBE, PIVOT, ROLLUP

OLAP Implementation (Cont.)

Consider again our retail shop example and the relation:

sales (*item name*, *color*, *clothes size*, *quantity*)

We can find the number of items sold in each item name by writing a simple

group by query:

select *item name*, **sum**(*quantity*)
from *sales*
group by *item name*;

The result of this query i

item_name	*quantity*
skirt	53
dress	35
shirt	49
pants	27

Query result

CUBE, PIVOT, ROLLUP OLAP Implementation (Cont.)

Similarly, we can find the number of items sold in each color, etc.

By using multiple attributes in the **group by** clause, we can find how many items were sold with a certain set of properties. For example, we can find a breakdown of sales by item-name and color by writing:

select *item name*, *color*, **sum**(*quantity*)
from *sales*
group by *item name*, *color*;

item_name	*color*	*quantity*
skirt	dark	8
skirt	pastel	35
skirt	white	10
dress	dark	20
dress	pastel	10
dress	white	5
shirt	dark	14
shirt	pastel	7
shirt	white	28
pants	dark	20
pants	pastel	2
pants	white	5

Figure 5.24 Query result.

CUBE, PIVOT, ROLLUP — OLAP Implementation (Cont.)

we want to generate the entire data cube using this approach, we would have to write a separate query for each of the following sets of attributes:

{ (*item name*, *color*, *clothes size*), (*item name*, *color*), (*item name*, *clothes size*), (*color*, *clothes size*), (*item name*), (*color*), (*clothes size*), () *}*

where () denotes an empty **group by** list.

The **cube** construct allows us to accomplish this in one query:

CUBE, PIVOT, ROLLUP

- The **cube** operation computes union of **group by**'s on every subset of the specified attributes
- E.g. consider the query

  select *item-name, color, size,* **sum**(*number*)
  from *sales*
  group by cube(*item-name, color, size*)

  This computes the union of eight different groupings of the *sales* relation:

  { (*item-name, color, size*), (*item-name, color*),
  (*item-name, size*), (*color, size*),
  (*item-name*), (*color*),
  (*size*), ( ) }

  where ( ) denotes an empty **group by** list.
- For each grouping, the result contains the null value for attributes not present in the grouping.

CUBE, PIVOT, ROLLUP Extended Aggregation (Cont.)

- Relational representation of cross-tab that we saw earlier, but with *null* in place of **all**, can be computed by

```
select item-name, color, sum(number)
from sales
group by cube(item-name, color)
```

- The function **grouping()** can be applied on an attribute
  - Returns 1 if the value is a null value representing all, and returns 0 in all other cases.

```
select item-name, color, size, sum(number),
    grouping(item-name) as item-name-flag,
    grouping(color) as color-flag,
    grouping(size) as size-flag,
from sales
group by cube(item-name, color, size)
```

- Can use the function **decode()** in the **select** clause to replace such nulls by a value such as **all**
  - E.g. replace *item-name* in first query by
    decode(**grouping**(item-*name*), 1, 'all', *item-name*)

CUBE, PIVOT, ROLLUP: Extended Aggregation (Cont.)

- The **rollup** construct generates union on every prefix of specified list of attributes
- E.g.

  select *item-name*, *color*, *size*, **sum**(*number*)
  from *sales*
  group by rollup(*item-name, color, size*)

  Generates union of four groupings:

  { (*item-name, color, size*), (*item-name, color*), (*item-name*), ( ) }
- Rollup can be used to generate aggregates at multiple levels of a hierarchy.
- E.g., suppose table *itemcategory*(*item-name, category*) gives the category of each item. Then

  select *category, item-name*, **sum**(*number*)
  from *sales, itemcategory*
  where *sales.item-name = itemcategory.item-name*
  group by rollup(*category, item-name*)

  would give a hierarchical summary by *item-name* and by *category.*

CUBE, PIVOT, ROLLUP Extended Aggregation (Cont.)

- Multiple rollups and cubes can be used in a single group by clause
  - Each generates set of group by lists, cross product of sets gives overall set of group by lists
- E.g.,

  select *item-name, color, size*, **sum**(*number*)
  from *sales*
  group by rollup(*item-name*), **rollup**(*color, size*)

  generates the groupings

  {item-name, ()} X {(color, size), (color), ()}

  = { (item-name, color, size), (item-name, color), (item-name), (color, size), (color), ( ) }

THANK YOU

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

shanthalapt@pes.edu

DATABASE MANAGEMENT SYSTEM

Dr.Geetha , Dr.Vinodha, Prof. Shanthala

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

PES UNIVERSITY

Specifying Constraints as Assertions and Actions as Triggers

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

DATABASE MANAGEMENT SYSTEM

Unit 3 : Specifying Constraints as Assertions and Actions as Triggers

PES UNIVERSITY

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

PES UNIVERSITY

T1 : CHAPTER 7.2

- Specifying Constraints as Assertions and Actions as Triggers

Database Management Systems

Specifying Constraints as Assertions and Actions as Triggers

PES UNIVERSITY

- Semantic Constraints: The following are beyond the scope of the EER and relational model
- **CREATE ASSERTION**
  - Specify additional types of constraints outside scope of built-in relational model constraints
- **CREATE TRIGGER**
  - Specify automatic actions that database system will perform when certain events and conditions occur

Database Management Systems

Specifying General Constraints as Assertions in SQL

- **CREATE ASSERTION**
  - Specify a query that selects any tuples that violate the desired condition
  - Use only in cases where it goes beyond a simple CHECK which applies to individual attributes and domains

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS   ( SELECT      *
                       FROM        EMPLOYEE E, EMPLOYEE M,
                                   DEPARTMENT D
                       WHERE       E.Salary>M.Salary
                                   AND E.Dno=D.Dnumber
                                   AND D.Mgr_ssn=M.Ssn ) );
```

Introduction to Triggers in SQL

- CREATE TRIGGER statement
  - Used to monitor the database
- Typical trigger has three components which make it a rule for an "active database "

  Event(s)

  Condition

  Action

USE OF TRIGGERS

- AN EXAMPLE with standard Syntax.(Note : other SQL implementations like PostgreSQL use a different syntax.)

R5:
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF Salary, Supervisor_ssn **ON** EMPLOYEE

FOR EACH ROW
WHEN (NEW.SALARY > **(** SELECT Salary FROM EMPLOYEE
WHERE Ssn = NEW. Supervisor_Ssn))
INFORM_SUPERVISOR (NEW.Supervisor.Ssn, New.Ssn)

THANK YOU

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Geetha, Dr.Vinodha, Prof.Shanthala

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Views and view implementation

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

PES UNIVERSITY

T1 : CHAPTER 7.3

Views and view implementation

DATABASE MANAGEMENT SYSTEM

Unit 3 : Views and view implementation

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

Database Management Systems

Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables called the defining tables
  - A view does not necessarily exists in physical form, in contrast to base tables whose tuples are actually stored in database.
  - This limits the possible update operations that can be applied to views but it does not provide any limitations on querying a view.

Database Management Systems

Specification of Views in SQL

- **CREATE VIEW** command
  - Give table name, list of attribute names, and a query to specify the contents of the view
  - In V1, attributes retain the names from base tables. In V2, attributes are assigned names

```
V1:   CREATE VIEW   WORKS_ON1
      AS SELECT     Fname, Lname, Pname, Hours
          FROM      EMPLOYEE, PROJECT, WORKS_ON
          WHERE     Ssn=Essn AND Pno=Pnumber;

V2:   CREATE VIEW   DEPT_INFO(Dept_name, No_of_emps, Total_sal)
      AS SELECT     Dname, COUNT (*), SUM (Salary)
          FROM      DEPARTMENT, EMPLOYEE
          WHERE     Dnumber=Dno
          GROUP BY  Dname;
```

Database Management Systems

Specification of Views in SQL (cont'd.)

- Once a View is defined, SQL queries can use the View relation in the FROM clause
- View is always up-to-date
  - Responsibility of the DBMS and not the user
- **DROP VIEW** command
  - Dispose of a view

Database Management Systems

View Implementation, View Update, and Inline Views

- The problem of efficiently implementing a view for querying is complex. Two main approaches are suggested :
- **Strategy1: Query modification** approach
  - This approach involves modifying the view query into query on underlying base tables.
  - Disadvantage: inefficient for views defined via complex queries that are time-consuming to execute

Strategy1: Query modification approach

```
QV1:    SELECT    Fname, Lname
        FROM      WORKS_ON1
        WHERE     Pname = 'ProductX';
```

For example, the query QV1 would be automatically modified to the following query by the DBMS:

```
SELECT    Fname, Lname
FROM      EMPLOYEE, PROJECT, WORKS_ON
WHERE     Ssn = Essn AND Pno = Pnumber
          AND Pname = 'ProductX';
```

Database Management Systems

View Materialization

- **Strategy 2: View materialization**
  - Physically create a temporary view table when the view is first queried
  - Keep that table on the assumption that other queries on the view will follow
  - Requires efficient strategy for automatically updating the view table when the base tables are updated
- **Incremental update strategy for materialized views**
  - DBMS determines what new tuples must be inserted, deleted, or modified in a materialized view table

Database Management Systems

View Materialization (contd.)

- Multiple ways to handle materialization:
  - **immediate update** strategy updates a view as soon as the base tables are changed
  - **lazy update** strategy updates the view when needed by a view query
  - **periodic update** strategy updates the view periodically (in the latter strategy, a view query may get a result that is not up-to-date). This is commonly used in Banks, Retail store operations, etc.

Database Management Systems

Views as authorization mechanism

- SQL query authorization statements (GRANT and REVOKE) are described in detail in Chapter 30
- Views can be used to hide certain attributes or tuples from unauthorized users
- E.g., For a user who is only allowed to see employee information for those who work for department 5, he may only access the view

DEPT5EMP:

```
CREATE VIEW  DEPT5EMP   AS
SELECT                  *
FROM                    EMPLOYEE
WHERE        Dno = 5;
```

Database Management Systems

Table 7.2 Summary of SQL Syntax

PES UNIVERSITY

Table 7.2 Summary of SQL Syntax

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                                { , <column name> <column type> [ <attribute constraint> ] }
                                [ <table constraint> { , <table constraint> } ] )
```

```
DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>
```

```
SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]
```

```
<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                        { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )
```

```
<grouping attributes> ::= <column name> { , <column name> }
```

```
<order> ::= ( ASC | DESC )
```

```
INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

Database Management Systems

Table 7.2 (continued) Summary of SQL Syntax

Table 7.2 Summary of SQL Syntax

DELETE FROM <table name> [WHERE <selection condition>]
UPDATE <table name> SET <column name> = <value expression> { , <column name> = <value expression> } [WHERE <selection condition>]
CREATE [UNIQUE] INDEX <index name> ON <table name> (<column name> [<order>] { , <column name> [<order>] }) [CLUSTER]
DROP INDEX <index name>
CREATE VIEW <view name> [(<column name> { , <column name> })] AS <select statement>
DROP VIEW <view name>

NOTE: The commands for creating and dropping indexes are not part of standard SQL.

Summary

- Complex SQL:
  - Nested queries, joined tables (in the FROM clause), outer joins, aggregate functions, grouping
- Handling semantic constraints with CREATE ASSERTION and CREATE TRIGGER
- CREATE VIEW statement and materialization strategies
- Schema Modification for the DBAs using ALTER TABLE , ADD and DROP COLUMN, ALTER CONSTRAINT etc.

THANK YOU

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Vinodha & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Programming Techniques

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

DATABASE MANAGEMENT SYSTEM

Unit 3 : Database Programming - functions,

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions
11. Database Programming - stored procedures
12. Database Programming - cursors

Database Programming - functions

In MYSQL we use the CREATE FUNCTION statement to create a new function that will be stored and this function can be further called by supplying any number of parameters and returning the required value.

Syntax:

The following is the syntax of CREATE FUNCTION statement –

```
DELIMITER $$
CREATE FUNCTION name_of_function(
parameter1,
parameter2,...
)
RETURNS datatype
[NOT] DETERMINISTIC
BEGIN
-- code of statements to be executed
END $$
DELIMITER ;
```

Database Programming - functions

- **name_of_ function –** It is the name of the function that needs to be created in MySQL.
- **parameter1, parameter2,… –** We can pass the optional parameters to the functions that need to be declared while creating it in the () brackets. A function can contain none, one or more than one parameter. These parameters can belong to either of the three types –
- **IN –** These types of parameters are assigned the values while calling the function and the value cannot be modified or overwritten inside the function but only referenced and used by the function.
- **OUT –** These are the parameters that can be assigned the values and overridden in the function but cannot be referenced by it.
- **IN OUT –** These types of parameters are assigned the values while calling the function and the value can be modified or overwritten inside the function as well as referenced and used by the function.
- **BEGIN and END –** BEGIN keyword marks the beginning of the function while END marks the completion of function in MYSQL.

Database Programming - functions

- **RETURN Datatype** – We can return any type of value from the execution of the function. The type of value that will be returned needs to be specified after the RETURN clause. Once, MySQL finds the RETURN statement while execution of the function, execution of the function is terminated and the value is returned.
- **DETERMINISTIC** – The function can be either deterministic or non-deterministic which needs to be specified here. When the function returns the same value for the same values of parameter then it is called deterministic. However, if the function returns a different value for the same values of functions then we can call that function to be nondeterministic. When none of the types of function is mentioned, then MySQL will consider function to be NON-DETERMINISTIC by default.

;

Database Programming - functions

- **Example of MySQL Create Function**

- ```
  DELIMITER $$
  CREATE FUNCTION isEligible(
  age INTEGER
  )
  RETURNS VARCHAR(20)
  DETERMINISTIC
  BEGIN
  IF age > 18 THEN
  RETURN ("yes");
  ELSE
  RETURN ("No");
  END IF;
  END$$
  DELIMITER
  ```

```
mysql> DELIMITER $$
mysql>
mysql> CREATE FUNCTION isEligible(
    -> age INTEGER
    -> )
    -> RETURNS VARCHAR(20)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE customerLevel VARCHAR(20);
    ->
    ->       IF age > 18 THEN
    -> RETURN ("yes");
    -> ELSE
    -> RETURN ("No");
    -> END IF;
    ->
    -> END$$
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
```
```

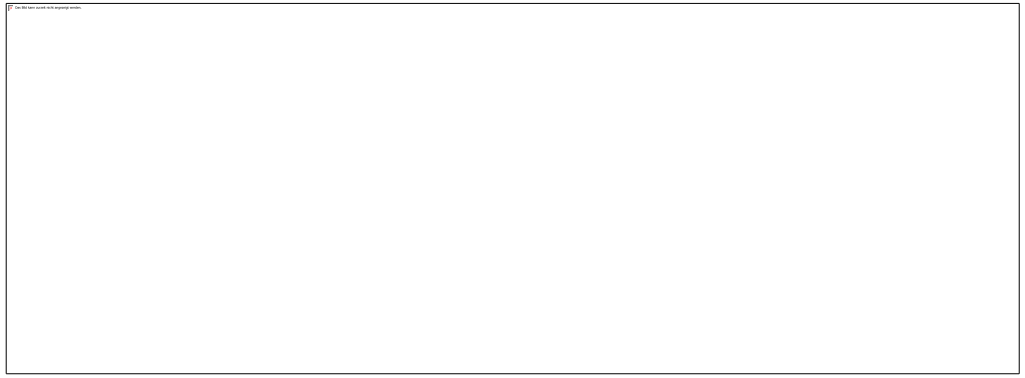
Database Programming - functions

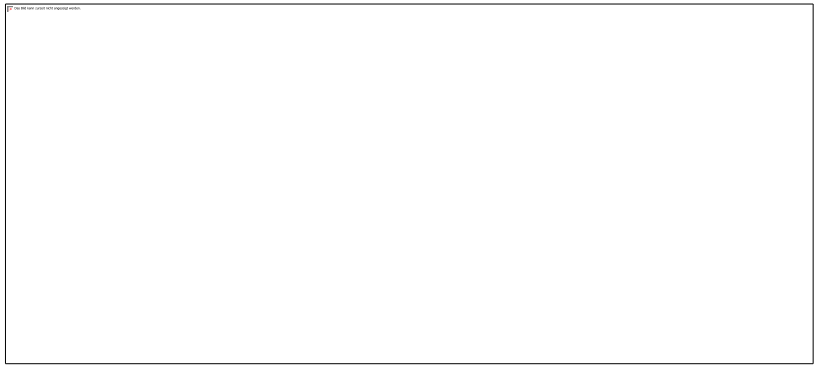
To check the eligibility for a 20-year-old guy call the function in the following way –

```
SELECT isEligible(20);
```

```
SELECT is eligible(10);
```

that results in the following output –



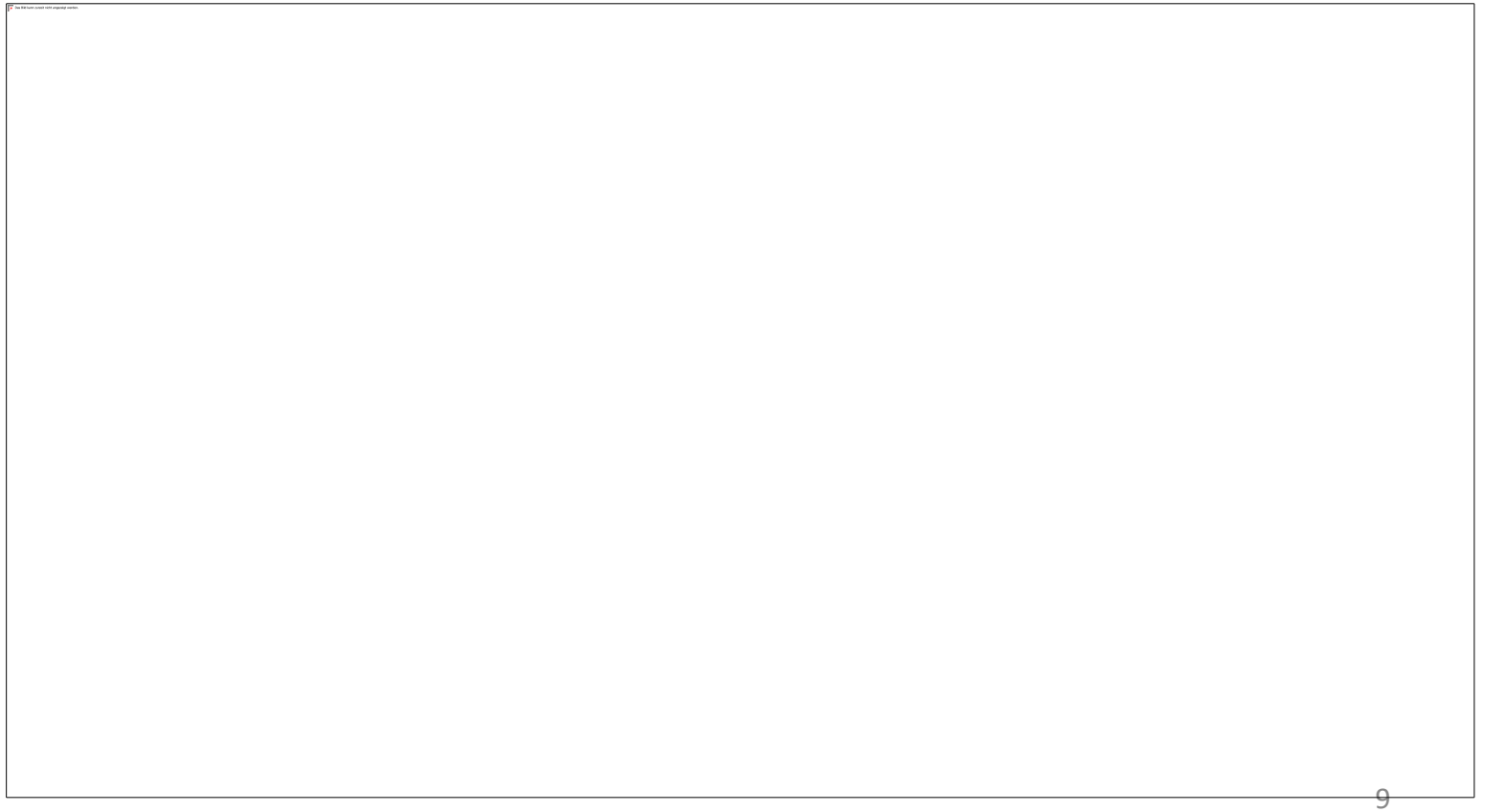


DATABASE MANAGEMENT SYSTEM

Database Programming - functions

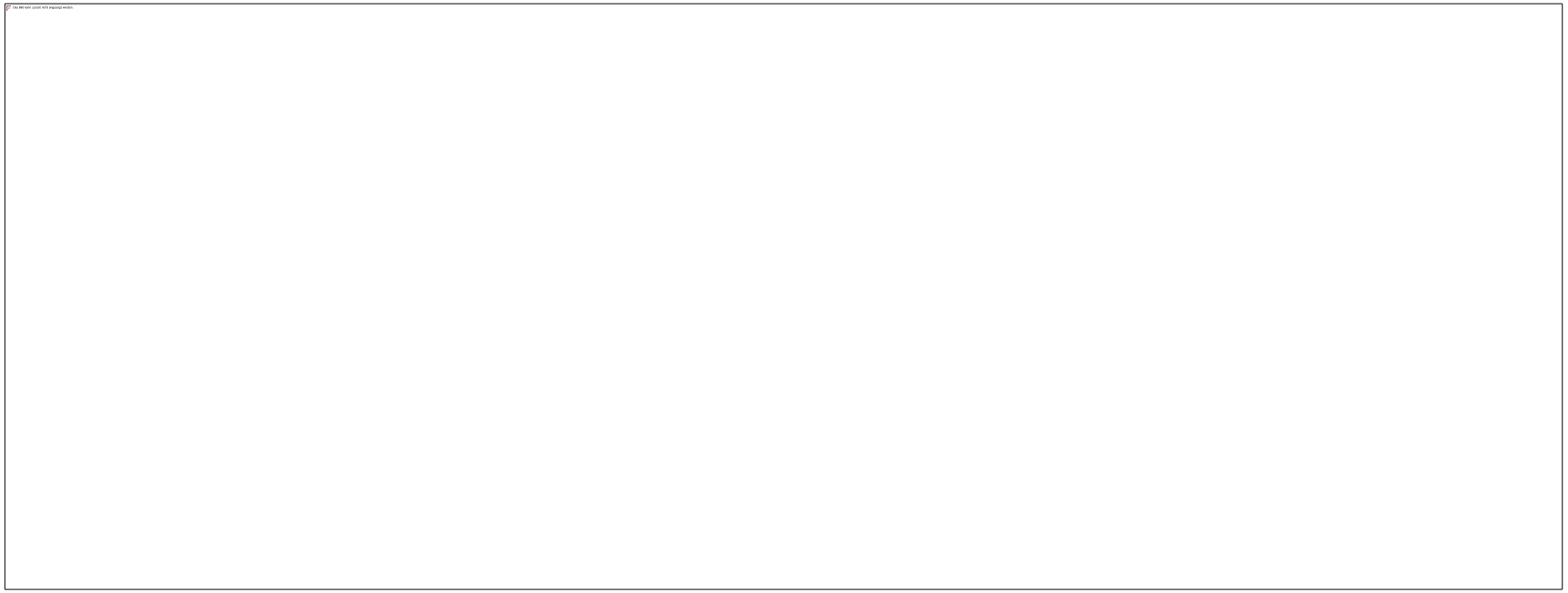
Example that involves returning the number of months between the current date and the supplied date

```
DELIMITER $$
CREATE FUNCTION getMonths(sampledate date) RETURNS int DETERMINISTIC
BEGIN
DECLARE currentDate DATE;
Select current_date()into currentDate;
RETURN (12 * (YEAR(currentDate)
- YEAR(sampledate))
+ (MONTH(currentDate)
- MONTH(sampledate)));
END
$$
DELIMITER ;
```



Database Programming - functions

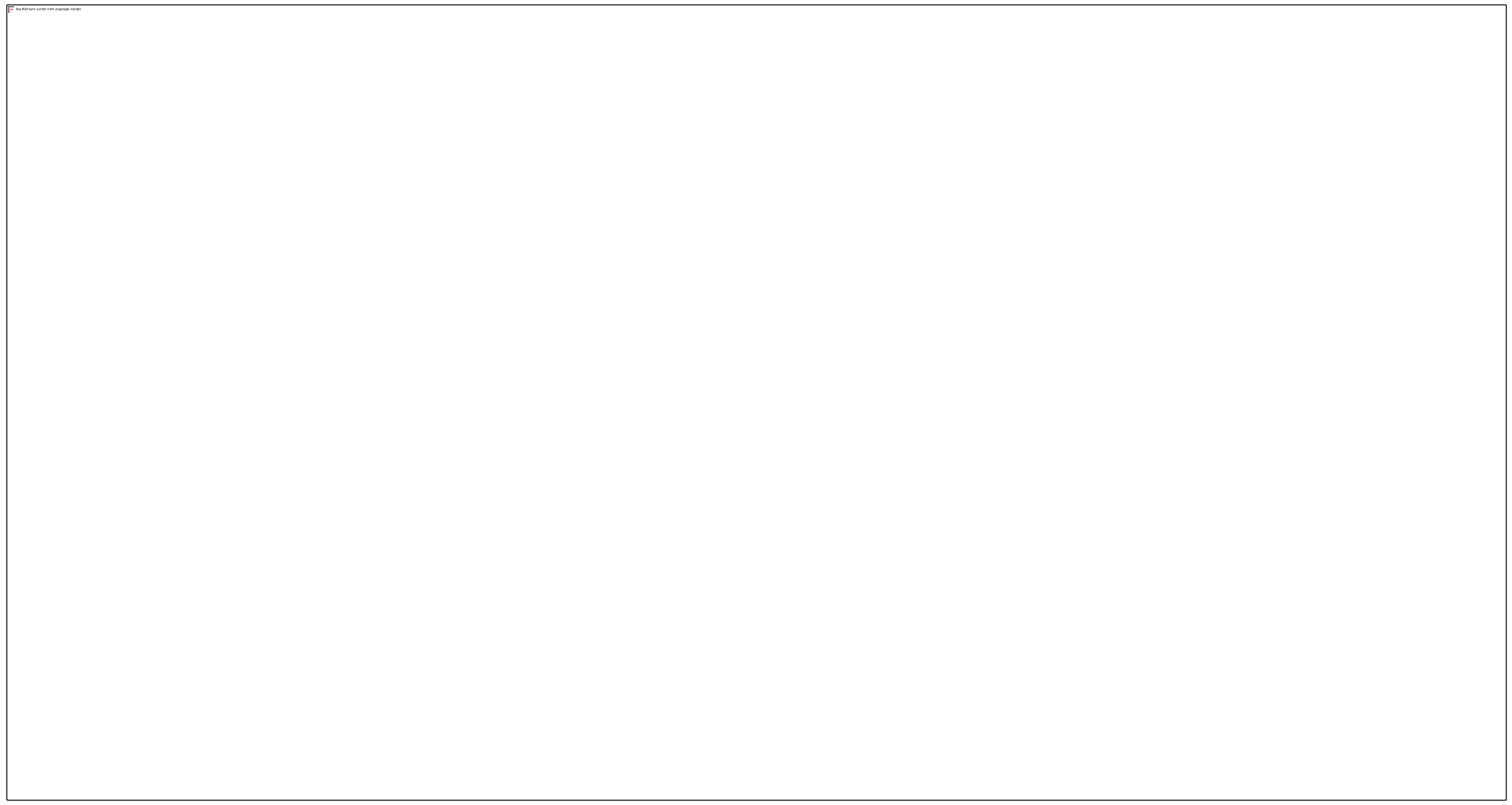
Consider the following Developer Table



Database Programming - functions

To retrieve the name of the developer and total months that he/she has worked in the company the following query statement is used in which a call to getMonths() function is invoked as follows–

SELECT name, getMonths(joining_date) as NumberOfMonths FROM developers;



THANK YOU

Dr.Vinodha & Dr. Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Programming Techniques

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Unit 3 : Database Programming - functions, stored procedures, cursors

1. SQL datatypes and advanced Blob and Clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - Functions
11. Database Programming - Stored procedures
12. Database Programming - Cursors

MySQL - STORED PROCEDURE

A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database. It is a subroutine or a subprogram in the regular computing language. A procedure always contains a name, parameter lists, and SQL statements. We can invoke the procedures by using triggers, other procedures and applications such as Java , Python , PHP , etc. The following syntax is used for creating a stored procedure in MySQL.

```
DELIMITER &&
CREATE PROCEDURE procedure_name [[IN | OUT | INOUT] parameter_name datatype [, parameter datatype]) ]
BEGIN
   Declaration_section
   Executable_section
END &&
DELIMITER ;
```

MySQL - STORED PROCEDURE

Modes of Procedure Parameters:

IN parameter

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

OUT parameters

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

INOUT parameters

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

How to call a stored procedure?

Syntax: CALL procedure_name (parameter(s))

MySQL - STORED PROCEDURE

Consider a table named **student_info** that contains the following data:

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM student_info;
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       1 | 101       | Mark      | English |    68 | 34545693537 |
|       2 | 102       | Joseph    | Physics |    70 | 98765435659 |
|       3 | 103       | John      | Maths   |    70 | 97653269756 |
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
|       5 | 105       | Rinky     | Maths   |    85 | 67531579757 |
|       6 | 106       | Adam      | Science |    92 | 79642256864 |
|       7 | 107       | Andrew    | Science |    83 | 56742437579 |
|       8 | 108       | Brayan    | Science |    85 | 75234165670 |
|      10 | 110       | Alexandar | Biology |    67 | 2347346438  |
+---------+-----------+-----------+---------+-------+-------------+
```

MySQL - STORED PROCEDURE

Procedure without Parameter:

To display all records of this table whose marks are greater than 70 and count all the table rows. The following code creates a procedure named get_merit_students:

```
DELIMITER &&
CREATE PROCEDURE get_merit_student ()
BEGIN
    SELECT * FROM student_info WHERE marks > 70;
    SELECT COUNT(stud_code) AS Total_Student FROM student_info;
END &&
DELIMITER ;
```

Let us call the procedure to verify the output:

```
mysql> CALL get_merit_student();
```

MySQL - STORED PROCEDURE

It will give the output as follows:

MySQL 8.0 Command Line Client

```
mysql> CALL get_merit_student();
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
|       5 | 105       | Rinky     | Maths   |    85 | 67531579757 |
|       6 | 106       | Adam      | Science |    92 | 79642256864 |
|       7 | 107       | Andrew    | Science |    83 | 56742437579 |
|       8 | 108       | Brayan    | Science |    85 | 75234165670 |
+---------+-----------+-----------+---------+-------+-------------+
5 rows in set (0.00 sec)

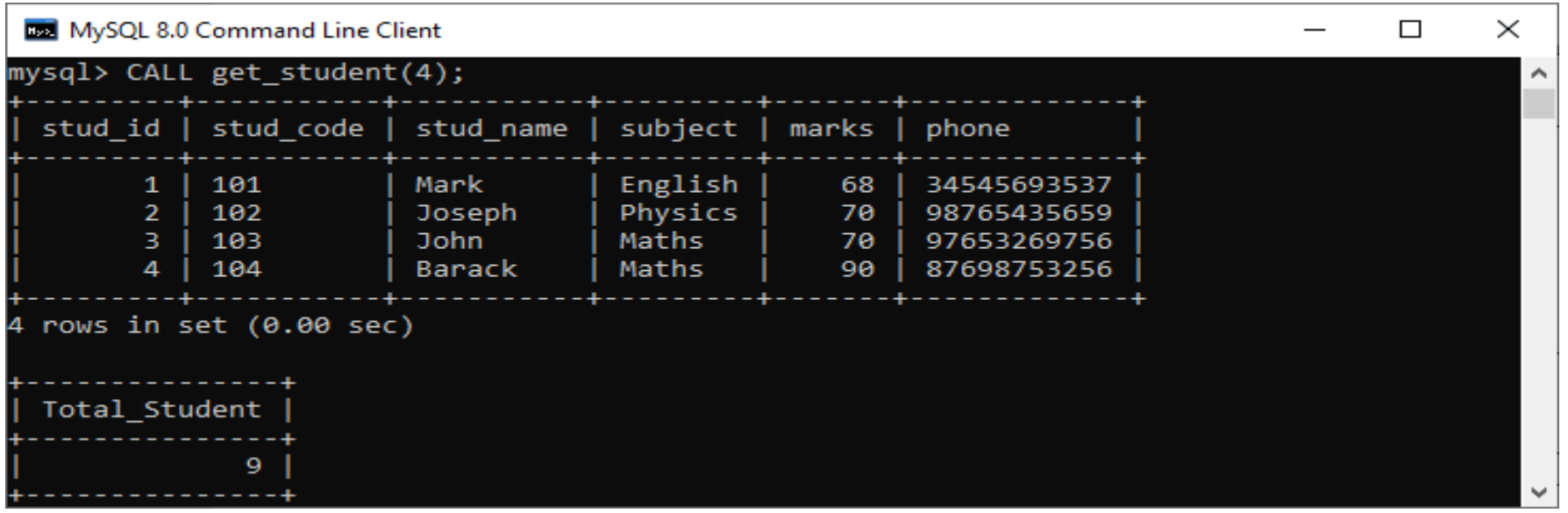
+---------------+
| Total_Student |
+---------------+
|             9 |
+---------------+
1 row in set (0.03 sec)
```

MySQL - STORED PROCEDURE

Procedures with IN Parameter:

```
DELIMITER &&
CREATE PROCEDURE get_student (IN var1 INT)
BEGIN
   SELECT * FROM student_info LIMIT var1;
   SELECT COUNT(stud_code) AS Total_Student FROM student_info;
END &&
DELIMITER ;
```

mysql> CALL get_student(4);



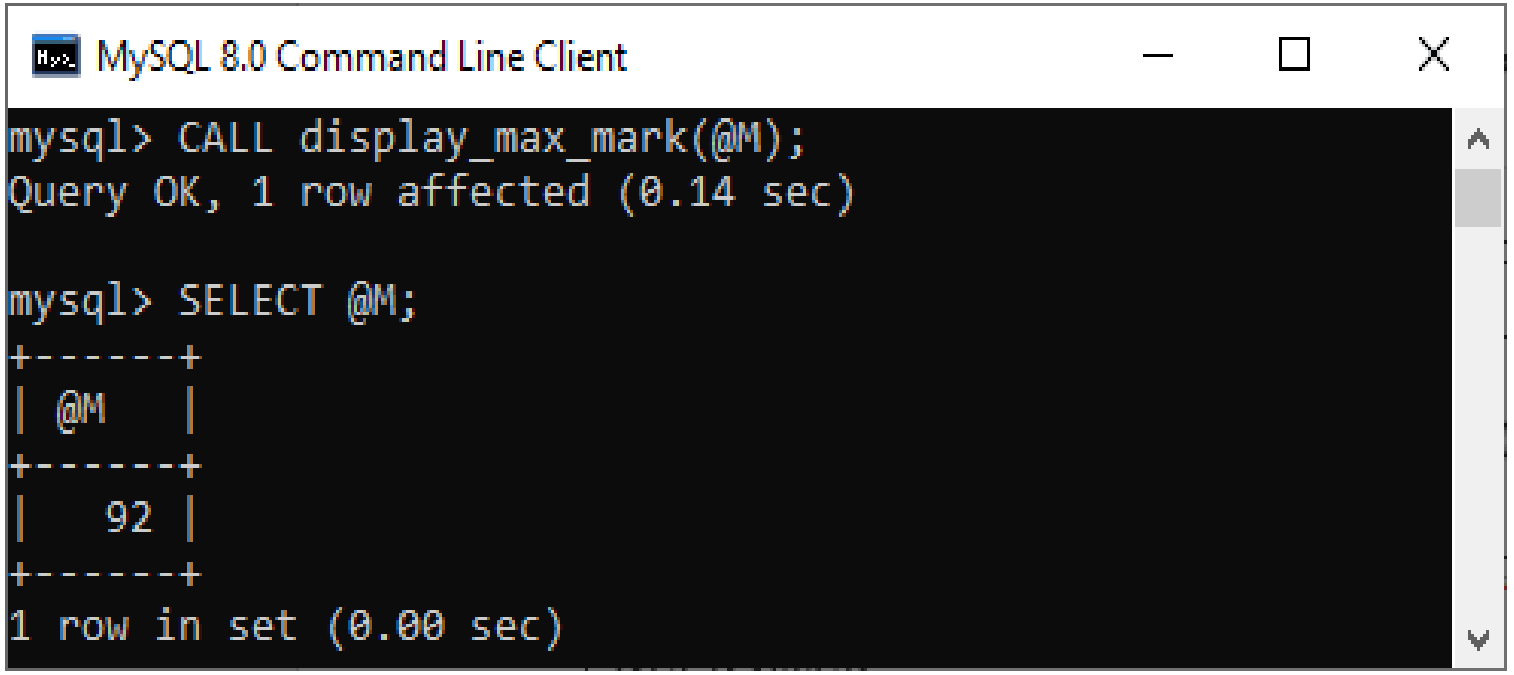
MySQL - STORED PROCEDURE

PES UNIVERSITY

Procedures with OUT Parameter:

```
DELIMITER &&
CREATE PROCEDURE display_max_mark (OUT highestmark INT)
BEGIN
   SELECT MAX(marks) INTO highestmark FROM student_info;
END &&
DELIMITER ;
```

mysql> CALL display_max_mark(@M);
mysql> SELECT @M;

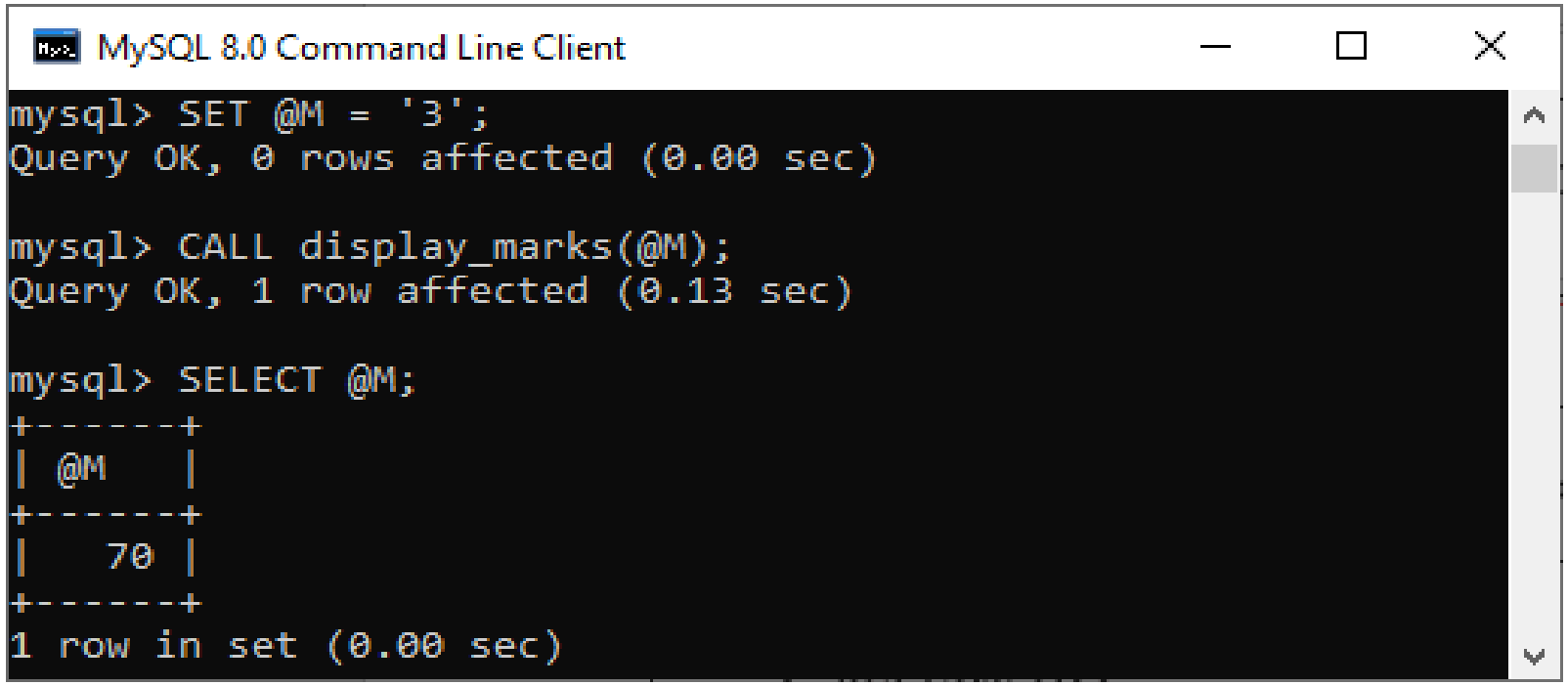


MySQL - STORED PROCEDURE

Procedures with INOUT Parameter:

```
DELIMITER &&
CREATE PROCEDURE display_marks (INOUT var1 INT)
BEGIN
   SELECT marks INTO var1 FROM student_info WHERE stud_id = var1;
END &&
DELIMITER ;
```

```
mysql> SET @M = '3';
mysql> CALL display_marks(@M);
mysql> SELECT @M;
```



MySQL - STORED PROCEDURE

How to delete/drop stored procedures in MySQL?

MySQL also allows a command to drop the procedure. When the procedure is dropped, it is removed from the database server also. The following statement is used to drop a stored procedure in MySQL:

DROP PROCEDURE [IF EXISTS] procedure_name;

Ex: mysql> DROP PROCEDURE display_marks;

We can verify it by listing the procedure in the specified database using the SHOW PROCEDURE STATUS command.

SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE search_condition]

Ex: mysql> SHOW PROCEDURE STATUS WHERE db = 'mystudentdb';

STORED PROCEDURE Vs FUNCTION

Stored Procedure	Function
Supports in, out and in-out parameters,i.e., input and output parameters	Supports only input parameters, no output parameters.
Stored procedures can call functions as needed	The function cannot call a stored procedure
There is no provision to call procedures from select/having and where statements	You can call functions from a select statement
Transactions can be used in stored procedures	No transactions are allowed
Can do exception handling by inserting try/catch blocks	No provision for explicit exception handling
Need not return any value	Must return a result or value to the caller
All the database operations like insert, update, delete can be performed	Only select is allowed

THANK YOU

Dr.Geetha & Dr.Vinodha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Vinodha K & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Programming Techniques

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Unit 3 : Views and view implementation

PES UNIVERSITY

1. SQL datatypes and advanced Blob and clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - functions, stored procedures, cursors
11. Database Programming - functions, stored procedures, cursors
12. Database Programming - functions, stored procedures, cursors

Cursors in MySQL

A database cursor is a mechanism that enables traversal over the records in a database. Cursors facilitate subsequent processing in conjunction with the traversal, such as retrieval, addition and removal of database records.

A cursor in SQL is a temporary work area created in system memory when a SQL statement is executed. A SQL cursor is a set of rows together with a pointer that identifies a current row. It is a database object to retrieve data from a result set one row at a time. It is useful when we want to manipulate the record of a table in a singleton method, in other words, one row at a time. In other words, a cursor can hold more than one row but can process only one row at a time. The set of rows the cursor holds is called the active set.

Cursors in MySQL

1. Implicit Cursors

Auto-created by Oracle when SQL is executed if there is no explicit cursor used.

Users or programmers cannot control the information or programs in it

Associated with INSERT, UPDATE and DELETE types of DML operation statements

Attributes: SQL%FOUND, SQL%NOTFOUND, %ISOPEN, %ROWCOUNT

2. Explicit Cursors

User-defined cursors which help to gain more control over the context part.

It is defined in the declaration area of the SQL block.

Created on SELECT statements that returns multiple records.

Attributes: SQL%FOUND, SQL%NOTFOUND, %ISOPEN, %ROWCOUNT.

Cursors

Properties:

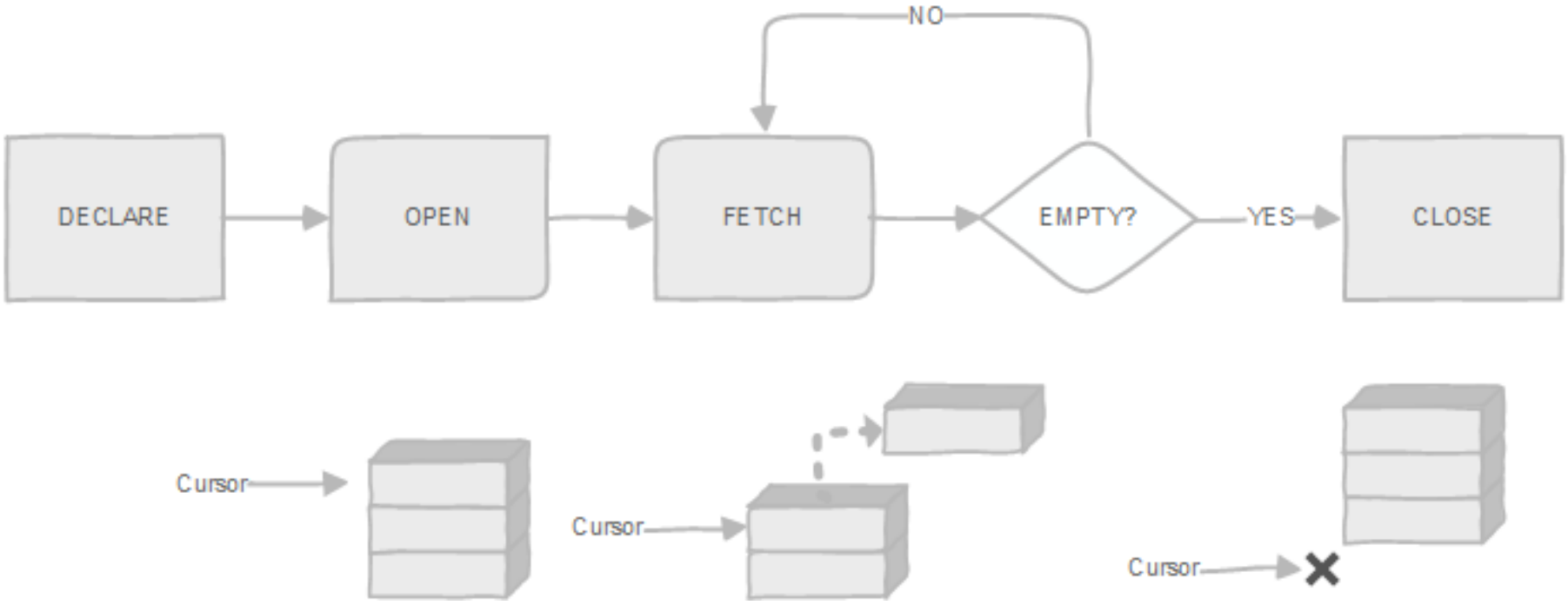
•READ ONLY − Using these cursors you cannot update any table.

•Non-Scrollable − Using these cursors you can retrieve records from a table in one direction i.e., from top to bottom.

•Asensitive − These cursors are insensitive to the changes that are made in the table i.e. the modifications done in the table are not reflected in the cursor, i.e. if we have created a cursor holding all the records in a table and, meanwhile if we add some more records to the table, these recent changes will not be reflected in the cursor we previously obtained.

Cursors

The following diagram illustrates how to use a cursor in MySQL:



Cursors

- First, declare a cursor.
- Next, open the cursor.
- Then, fetch rows from the result set into a target.
- After that, check if there is more row left to fetch. If yes, go to step 3, otherwise, go to step 5.
- Finally, close the cursor.

Cursors

Syntax

```
DECLARE cursor_name CURSOR FOR select_statement
OPEN cursor_name
 FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
CLOSE cursor_name
```

Cursors

When the FETCH statement is called, the next row is read in the result set each time. But a time comes when it reaches to the end of the set and no data is found there, so to handle this condition with MYSQL cursor we need to use a NOT FOUND handler.

Syntax:
DECLARE CONTINUE HANDLER FOR NOT FOUND SET variable_name = 1;

Attribute	Description
%FOUND	It will return TRUE in case an INSERT, UPDATE, or DELETE statement affects one or more rows or a SELECT INTO statement returns one or more rows. In other cases, it will return FALSE.
%NOTFOUND	It is technically the opposite of %FOUND attribute. It returns TRUE in case an INSERT, UPDATE, or DELETE statement doesn't affect any rows or a SELECT INTO statement returns no rows. Else it returns just FALSE.
%ISOPEN	This attribute will always return FALSE for implicit cursors as the SQL cursor is automatically closed immediately after the associated SQL statement is executed.
%ROWCOUNT	It returns the total number of affected rows by an INSERT, UPDATE, or DELETE statement, or the rows returned by a SELECT INTO statement.

Cursors

Example: Consider the following Tutorials table

ID	TITLE	AUTHOR	DATE
1	Java	Krishna	2019-09-01
2	JFreeCharts	Satish	2019-05-01
3	JavaSprings	Amit	2019-05-01
4	Android	Ram	2019-03-01
5	Cassandra	Pruthvi	2019-04-06

Cursors

create another table to back up the data −

```
CREATE TABLE backup (
 ID INT, TITLE VARCHAR(100),
AUTHOR VARCHAR(40),
DATE VARCHAR(40)
);
```

Write procedure that backups the contents of the tutorials table to the backup table using cursors −

Cursors

```
DELIMITER //
CREATE PROCEDURE ExampleProc()
   BEGIN
      DECLARE done INT DEFAULT 0;
      DECLARE tutorialID INTEGER;
      DECLARE tutorialTitle, tutorialAuthor, tutorialDate VARCHAR(20);
      DECLARE cur CURSOR FOR SELECT * FROM tutorials;
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
      OPEN cur;
      label: LOOP
      FETCH cur INTO tutorialID, tutorialTitle, tutorialAuthor, tutorialDate;
      INSERT INTO backup VALUES(tutorialID, tutorialTitle, tutorialAuthor, tutorialDate);
      IF done = 1 THEN LEAVE label;
      END IF;
      END LOOP;
      CLOSE cur;
   END//
DELIMITER ;
```

Cursors

call the above procedure as shown below −

```
Mysql> CALL ExampleProc;
Query OK, 1 row affected (0.78 sec)
```

Next, verify the contents of the backup table −

```
mysql> select * from backup;
+------+-------------+---------+------------+
| ID   | TITLE       | AUTHOR  | DATE       |
+------+-------------+---------+------------+
| 1    | Java        | Krishna | 2019-09-01 |
| 2    | JFreeCharts | Satish  | 2019-05-01 |
| 3    | JavaSprings | Amit    | 2019-05-01 |
| 4    | Android     | Ram     | 2019-03-01 |
| 5    | Cassandra   | Pruthvi | 2019-04-06 |
+------+-------------+---------+------------+
5 rows in set (0.08 sec)
```

Cursors

Consider the following table of employees. Using the stored procedure and cursors concatenate name and salary of the employees

```
mysql> select * from employees;
+-----+----------+--------+--------+-------------+-------+------------+--------+------------+
| eid | ename    | salary | deptid | designation | mgrid | commission | gender | hiredate   |
+-----+----------+--------+--------+-------------+-------+------------+--------+------------+
|   1 | nikhil   | 180000 |     10 | Manager     |  NULL |      30000 | m      | 2015-02-05 |
|   2 | nitin    | 140000 |     10 | Manager     |  NULL |      15000 | m      | 2013-06-21 |
|   3 | sonali   |  90000 |     20 | Developer   |     1 |       5000 | f      | 2017-03-05 |
|   4 | anshita  | 100000 |     20 | Analyst     |     2 |       NULL | f      | 2017-01-25 |
|   5 | jayant   |  50000 |     30 | Analyst     |     2 |       2000 | m      | 2017-03-18 |
|   6 | abhinav  | 110000 |     20 | Trainer     |     1 |       4000 | m      | 2016-04-23 |
|   7 | vaibhav  |  60000 |     30 | Developer   |     1 |       NULL | m      | 2016-08-12 |
|   8 | rashmita |  55000 |     20 | HRhead      |     1 |       6000 | f      | 2015-04-15 |
|   9 | neha     | 107000 |     20 | Trainer     |     2 |      12000 | f      | 2013-03-19 |
|  10 | rajnish  |  80000 |     20 | Technician  |     1 |          0 | m      | 2013-05-27 |
|  11 | wasim    |  54000 |     30 | Assitant    |     1 |       NULL | m      | 2017-03-29 |
+-----+----------+--------+--------+-------------+-------+------------+--------+------------+
```

```
delimiter $$
create procedure proc_emp()
begin
    declare v_ename varchar(100);
    declare v_salary int;
    declare v_finished integer default 0;
    declare c1 cursor for select ename , salary from employees;
    declare continue handler for NOT FOUND set v_finished=1;
    open c1;
    get_emp: LOOP
        fetch c1 into v_ename , v_salary;
        if v_finished=1 then
            leave get_emp;
        end if;
        select concat(v_ename,v_salary);
   END LOOP get_emp;
   close c1;
end $$
```

THANK YOU

Dr.Vinodha K & Dr.Geetha

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Dr.Geetha & Dr.Vinodha K

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Programming Techniques

Slides adapted from Author Slides of Fundamentals of Database Systems", Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Department of Computer Science and Engineering

DATABASE MANAGEMENT SYSTEM

Unit 3 : Database Programming - functions, stored procedures, cursors

PES UNIVERSITY

1. SQL datatypes and advanced Blob and Clob datatypes
2. Constraints and SQL Commands, schema changes in SQL
3. Advanced SQL Queries, Nested Queries
4. Sub- queries, Correlated sub-queries
5. Outer join, Aggregation function and group, having clause
6. Other SQL Constructs: WITH, CASE
7. CUBE, PIVOT, ROLLUP
8. Specifying General Constraints as Assertions and Triggers
9. Views and view implementation
10. Database Programming - Functions
11. Database Programming - Stored procedures
12. Database Programming - Cursors

Python with MySQL Database Connection

Steps to connect a python application to database:

- Import mysql.connector module
- Create the connection object.
- Create the cursor object
- Execute the query

How to Connect to MySQL Database in Python

1. Install MySQL connector module

Use the pip command to install MySQL connector Python.

pip install mysql-connector-python

Python with MySQL Database Connection

2. Import MySQL connector module
Import using a import mysql.connector statement so you can use this module's methods to communicate with the MySQL database.

3. Use the connect() method
Use the connect() method of the MySQL Connector class with the required arguments to connect MySQL. It would return a MySQLConnection object if the connection established successfully

4. **Use the cursor() method**
Use the cursor() method of a MySQLConnection object to create a cursor object to perform various SQL operations.

5. Use the execute() method
The execute() methods run the SQL query and return the result.

Python with MySQL Database Connection

6. Extract result using fetchall()

Use cursor.fetchall() or fetchone() or fetchmany() to read query result.

7. Close cursor and connection objects

Use cursor.clsoe() and connection.clsoe() method to close open connections after your work completes

Python with MySQL Database Connection

PES UNIVERSITY

Example to connect to MySQL Database in Python

```
import mysql.connector
from mysql.connector import Error

try:
    conn = mysql.connector.connect(host='localhost',
                                   database='Electronics',
                                   user='root',
                                   password='')
    if conn.is_connected():
            cursor=conn.cursor()
            cursor.execute("drop table if exists empdetails");
            cursor.execute("create table empdetails(empno int,empname varchar(20))")
            print"**Table created**"
            cursor.execute("insert into empdetails values(1,'Reena')")
            cursor.execute("insert into empdetails values(2,'Reetta')")
            cursor.execute("insert into empdetails values(3,'Rekha')")
```

Python with MySQL Database Connection

```
print"**3 Records Inserted**"
cursor.execute("select * from empdetails");
print"***Records Retrieved**"
while(1):
        row=cursor.fetchone()
        if row==None:
                break
        print"%s %s"%(row[0],row[1])
print"The number of rows returned %d"%cursor.rowcount
except Error as e:
    print("Error while connecting to MySQL", e)
finally:
    if conn.is_connected():
        cursor.close()
        conn.close()
        print("MySQL connection is closed")
```

Output:

```
**table created**
**3 records inserted**
*** records retrieved***
1 Reena
2 Reeta
3 Rekha
The number of rows returned 3
```

THANK YOU

Dr.Geetha & Dr.Vinodha K

Department of Computer Science and Engineering