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
Example of WITH

• See an alternate approach to doing 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;

Retrieve the department number having more than 5 employees and the number of its employees who are making more than $40,000
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

- U6': UPDATE EMPLOYEE
  SET Salary = CASE
    WHEN Dno = 5 THEN Salary + 2000
    WHEN Dno = 4 THEN Salary + 1500
    WHEN Dno = 1 THEN Salary + 3000
  END
Constraint
Entity Integrity and Referential Integrity

- **Entity integrity constraint** states that no primary key value can be NULL.

- **Referential integrity constraint** states that every value of a foreign key must match a values of an existing primary key.
Assertions and Triggers

• Specifying
  – Constraints as Assertions
  – Actions as Triggers

• **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
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

```sql
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**
CREATE TRIGGER upd_check BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
  IF NEW.amount < 0 THEN
    SET NEW.amount = 0;
  ELSEIF NEW.amount > 100 THEN
    SET NEW.amount = 100;
  END IF;
END;
/*
delimiter ;
Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables called the defining tables
  - Considered to be a virtual table that is not necessarily populated
• **CREATE VIEW** command
  
  – Give table name, list of attribute names, and a query to specify the contents of the view
  
  – In $V_1$, attributes retain the names from base tables. In $V_2$, attributes are assigned names

\[
V_1:\quad \text{CREATE VIEW } \text{WORKS\_ON1} \\
\quad \text{AS SELECT } \text{Fname, Lname, Pname, Hours} \\
\quad \text{FROM } \text{EMPLOYEE, PROJECT, WORKS\_ON} \\
\quad \text{WHERE } \text{Ssn=Essn AND Pno=Phnumber;} \\
\]

\[
V_2:\quad \text{CREATE VIEW } \text{DEPT\_INFO(Dept\_name, No\_of\_emps, Total\_sal)} \\
\quad \text{AS SELECT } \text{Dname, COUNT (*), SUM (Salary)} \\
\quad \text{FROM } \text{DEPARTMENT, EMPLOYEE} \\
\quad \text{WHERE } \text{Dnumber=Dno} \\
\quad \text{GROUP BY } \text{Dname;} \\
\]
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
View Implementation, View Update, and Inline Views

• Complex problem of efficiently implementing a view for querying

• **Strategy 1: Query modification** approach
  – Compute the view as and when needed. Do not store permanently
  – Modify view query into a query on underlying base tables
  – Disadvantage:
    • inefficient for views defined via complex queries that are time-consuming to execute
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
• 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.
View Update

- Update on a view defined on a single table without any aggregate functions
  - Can be mapped to an update on underlying base table if the primary key is preserved in the view
- Update not permitted on aggregate views. E.g.,

```
UV2: UPDATE DEPT_INFO
    SET Total_sal = 100000
    WHERE Dname = 'Research';
```

cannot be processed because Total_sal is a computed value in the view definition
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:

  ```sql
  CREATE VIEW DEPT5EMP AS
  SELECT * 
  FROM EMPLOYEE 
  WHERE Dno = 5;
  ```
• 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 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 COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);`
Adding and Dropping Constraints

• Change constraints specified on a table
  – Add or drop a named constraint

  – `ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT EMPSUPERFK CASCADE;`
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.

```sql
ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;
```

• Default values can be dropped and altered:

```sql
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn DROP DEFAULT;
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn SET DEFAULT ‘333445555’;
```
EXPANDED Block Structure of SQL Queries

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```
## Differences between Database and Tables

<table>
<thead>
<tr>
<th>Operation</th>
<th>Database</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>CREATE TABLE</td>
<td>INSERT INTO &lt;TB&gt;</td>
</tr>
<tr>
<td>Remove</td>
<td>DROP TABLE</td>
<td>DELETER FROM &lt;TB&gt;</td>
</tr>
<tr>
<td>Modify</td>
<td>ALTER TABLE</td>
<td>UPDATE &lt;TB&gt; SET</td>
</tr>
</tbody>
</table>
• Some of the slides in this presentation are taken from the slides provided by the authors.
• Many of these slides are taken from cs145 course offered by Stanford University.
• Thanks to YouTube, especially to Dr. Daniel Soper for his useful videos.