# CS3402: Chapter 3
# SQL: Structured Query Language

# SQL: Structured Query Language

- SQL is a hybrid language, its basically 4 types of languages in one
  - Data Definition Language(DDL)
    used for define database schemas
  - Data Query Language (DQL)
    used to query the database for information
  - Data Manipulation Language (DML)
    used for inserting, updating and deleting data from the database
  - Data Control Language (DCL)
    used for controlling the access to the data in the database

- Each statement in SQL ends with a semicolon (;)
- SQL is case insensitive
- Implementations of SQL in Different DBMS (MySQL, Oracle) are slightly different

# Data Definition Language(DDL)

# CREATE Table

- Create Database:
  - `CREATE SCHEMA COMPANY;`
- Create TABLE:
  - `CREATE TABLE EMPLOYEE …`
- Base tables (base relations)
  - Relation and its tuples are actually (physically) created and stored as a file by the DBMS
  - The tables are stored in the secondary storage in the specified format
- Virtual relations
  - Created through the `CREATE VIEW` statement
  - The tables are not actually created but are presented to the user through reconstruction (view) of base tables
  - View is always up-to-date (change in base table will be reflected in the views automatically)

# Table Manipulation

- Table creation: CREATE TABLE table_name
  (column_name1 data_type(size),
  column_name2 data_type(size),....);
  CREATE VIEW view_name AS SELECT .... FROM...

- Table deletion: DROP TABLE table-name;
  DROP VIEW view-name

- Table update:
  ALTER TABLE table-name
  ADD Aj, Dj
  (to add new attribute Aj with domain Dj to an existing table)
  ALTER VUEW view-name

# One possible database state for the COMPANY relational database schema

**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

**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

```
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) );
```

Indicate this column does not accept NULL value

Define primary key

ensures that all values in a column are different

Define foreign key and its reference

# SQL CREATE TABLE data definition statements for defining the COMPANY schema

```
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) );
```

# Data Query Language(DQL)

# The SELECT-FROM-WHERE Structure of Basic SQL Queries

- SELECT statement
  - The basic statement for retrieving information from a database

**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.

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

**Specified attributes** ↓

Satisfy the conditions →

| Title | Year | Length | Type |
|---|---|---|---|
| Star War | 1977 | 124 | Color |
| Mighty Duck | 1991 | 104 | Color |
| Wayne's World | 1992 | 95 | Color |

# Retrieval from a single table

**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';
```

← Projection attributes

← Selection conditions

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

- Logic operators to connect multiple conditions: **Not, AND, OR**
- Priority: (), NOT, AND OR
- E.g. Lname = 'Lee' OR Lname = 'Smith' AND Sex = 'Male"
  (Lname = 'Lee' OR Lname = 'Smith') AND Sex = 'Male"

# Retrieval from two tables

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

```
Q1:   SELECT   Fname, Lname, Address
      FROM     EMPLOYEE, DEPARTMENT      <-- Two tables
      WHERE    Dname='Research' AND Dnumber=Dno;   <-- Join conditions
```

| 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 |

# Retrieval from two tables:Join Operation

| A | B |
|---|---|
| ABC | DEF |
| | |

X

JOIN

| C | D | E | F |
|---|---|---|---|
| XYZ | | | |
| AAAA | | | |
| BBBB | | | |
| CCCC | | | |

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| ABC | DEF | XYZ | | | |
| ABC | DEF | AAAA | | | |
| ABC | DEF | BBBB | | | |
| ABC | DEF | CCCCC | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# Retrieval from two tables

(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';
```

Select-project-join query

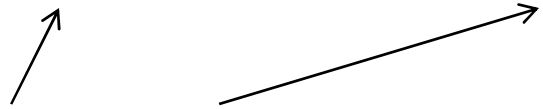
Project joins Department

Department joins Employee

# 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;
```



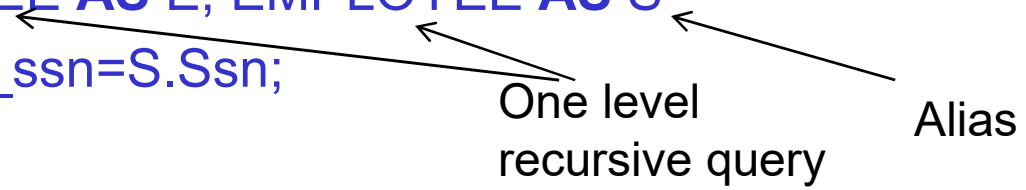
Dnumber

# 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, and Renaming

- The attribute names can also be renamed

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

- The “AS” may be dropped in most SQL implementations

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

# DISTINCT, LIKE

- *Find the first name of the employees:*

```
SELECT DISTINCT Fname
    FROM EMPOYEE;
```

Records with same values will be removed

- *Find the names of all employees whose first name has the substring 'mm' included*

```
SELECT Fname, Minit, Lname
 FROM EMPOYEE
 WHERE Fname LIKE "%mm%";
```

(Note: if we use " _ _ mm%", then it becomes a special case)

3rd character (case sensitive)

# Ordering Tuples

- List first name of all employees in alphabetic order

```
SELECT DISTINCT Fname
 FROM EMPOYEE
 ORDER BY Fname;
```

*By default, in ascending order.*

- List the employee names in descending order of last name, and if several employees have the same last name, order them in ascending order by the first name

```
SELECT Fname, Minit, Lname
 FROM EMPLOYEE
 ORDER BY Lname DESC, Fname ASC;
```

# Unspecified WHERE Clause and Use of the Asterisk

- Missing `WHERE` clause
  - Indicates no condition on tuple selection (select ALL)
- Effect is a `CROSS PRODUCT (JOIN n x m)`
  - Result is all possible tuple combinations result

**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;
```

May return many tuples
Join operation

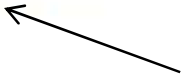
# Unspecified WHERE Clause and Use of the Asterisk

- Specify an asterisk (*)
  - Retrieve all the attribute values of the selected tuples

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

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

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



EMPLOYEE: 100 tuples
DEPARTMENT: 10 tuples
Finally, 1000 tuples and all attributes

# Tables as Sets in SQL

- SQL does not automatically eliminate duplicate tuples (the attributes of two tuples have same values) in query results (NOT a set)
- 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`, `INTERSECT`, `MINUS/EXCEPT` (difference),

**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' );
```

Same attribute

The projects in the dept with "Smith" as manager

# Tables as Sets in SQL, UNION

- Assuming the following relation schemes:

  Customer (cname, street, city)

  Branch (bname, assets, b-city)

  Borrow (bname, loan#, cname, amount)

  Deposit (bname, acct#, cname, balance)

  - *Find all customers of the Kowloon branch*

```
(SELECT cname
 FROM Deposit
 WHERE bname = “Kowloon”)
```

A set of tuples

```
UNION
(SELECT cname
 FROM Borrow
 WHERE bname = “Kowloon”);
```

A set of tuples

# Tables as Sets in SQL, MINUS/EXCEPT

- Assuming the following relation schemes:

  Customer (cname, street, city)

  Branch (bname, assets, b-city)

  Borrow (bname, loan#, cname, amount)

  Deposit (bname, acct#, cname, balance)

  - *Find all customers of who has deposit account in Kowloon branch but no loan account in Kowloon branch*

```
(SELECT cname
 FROM Deposit
 WHERE bname = "Kowloon")
```

A set of tuples

```
MINUS
(SELECT cname
 FROM Borrow
 WHERE bname = "Kowloon");
```

A set of tuples

# Nested Queries and Set / Multiset Comparisons

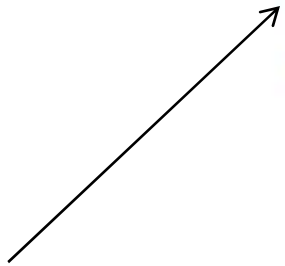
- Nested queries
  - select-from-where blocks within WHERE clause of another query
  - E.g., some queries require that existing values in the database be fetched and then used in a comparison condition

- 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

# Nested Queries, IN

- 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' );
```



Select the Essn of all employees who work the same (project, hours) as the employee Essn ="123456789"

# Nested Query, IN

- Find all customers who have an account at some branch in which Jones has an account

**solution 1:**

```
SELECT DISTINCT T.cname
 FROM Deposit T
 WHERE T.cname != "Jones"
     AND T.bname IN (SELECT S.bname
                     FROM Deposit S
                     WHERE S.cname = "Jones");
```

Deposit = T

Deposit = S

A set of records with "Jones" as cname

**solution 2:**

```
SELECT DISTINCT T.cname
 FROM Deposit S, Deposit T
 WHERE S.cname = "Jones"  AND S.bname = T.bname
     AND T.cname != S.cname;
```

# Nested Query, EXISTS

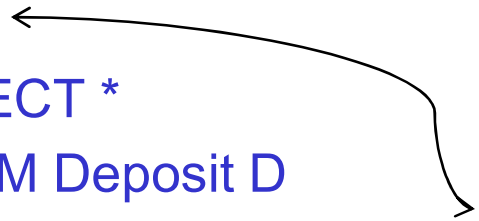
- `EXISTS` function
  - Check whether the result of a nested query is empty or not
  - Typically used in conjunction with a correlated nested query

- Correlated nested query
  - Evaluated once for each tuple in the outer query
  - Whenever a condition in the WHERE clause of a nested query references some attributes of a relation declared in the outer query

# Nested Query, EXISTS

- Find all customers of Central branch who have an account there but no loan there

```
SELECT C.cname
 FROM Customer C
 WHERE EXISTS (SELECT *
                 FROM Deposit D
                 WHERE D.cname = C.cname
                     AND D.bname = “Central”)
AND   NOT EXISTS
                (SELECT *
                 FROM Borrow B
                 WHERE B.cname = C.cname
                     AND B.bname = “Central”);
```



# Difference between EXISTS and IN

Query the record in t1 which has different value to any of the record in t2

**solution 1:**

```
select *  from table1 t1
where not exists (select * from table2 t2
where t2.value = t1.value);
```

**solution 2:**

```
select *
from table1 t1
where t1.value not in (select t2.value
from table2 t2);
```

Q1：Solution 1 and 2 are equivalent or not？

Q2：Which solution is more efficient？

# Nested Queries, SOME/ANY, ALL

- Comparison operators that can be combined with ANY (or SOME) ALL: = , >, >=, <, <=, and <>
  - **ANY** (or **SOME**) operator
    - The **ANY** operator returns true if any of the subquery values meet the condition.
  - **ALL** operator
    - The **ALL** operator returns true if all of the subquery values meet the condition.

# Nested Queries, ALL

- Find names of all employees that have more salary than all employees in the department with department no. 5

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

The salary of Employee tuples with Dno = 5

# Nested Query, ALL

- Find branches having greater assets than all branches in N.T.

***solution 1:***

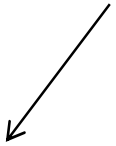
```
SELECT bname
 FROM Branch
 WHERE assets > ALL (SELECT assets
                     FROM Branch
                     WHERE b-city = "New Territory");
```

The branches in N.T.

***solution 2:***

```
SELECT X.bname
 FROM Branch X
 WHERE NOT EXISTS (SELECT *
                   FROM Branch Y
                   WHERE Y.b-city = "New Territory"
                     AND Y.assets >= X.assets);
```

The branches in N.T. with assets greater than branch X



# Nested Queries, SOME/ANY

- Find names of all branches that have greater assets than some branch located in Central

**solution 1:**

```
SELECT bname
 FROM Branch
 WHERE assets > SOME (SELECT assets
                      FROM Branch
                      WHERE b-city = “Central”);
```

The assets of the branches in “Central”

Anyone

**solution 2:**

```
SELECT X.bname
 FROM Branch X, Branch Y
 WHERE X. assets > Y.assets
      AND Y.b-city= “Central”;
```

# Aggregate Functions

- Used to summarize information from multiple tuples into a single-tuple

- 1 Grouping （**GROUP BY**）
  - Create subgroups of tuples before summarizing
  - E.g., A group of tuples => Count its number to be the value of a new attribute
- 2 Built-in aggregate functions
  - `COUNT`, `SUM`, `MAX`, `MIN`, and `AVG`
  - Functions can be used in the SELECT clause or in a HAVING clause
- 3 Having：

  filtering subgroups with conditions

# Aggregate Functions

- Examples:
  - assume the following relational schemes:
    Borrow (b-name, loan#, c-name, amount)
    Customer (c-name, street, city)
    Branch (b-name, assets, b-city)
    Deposit (b-name, acct#, c-name, balance)

# GROUP BY

- `GROUP BY` clause
  - Provides a condition to select or reject an entire group:
  - Find the average account balance at each branch

```
SELECT b-name, AVG(balance)
FROM Deposit
GROUP BY b-name;
```

← AVG(balance) is the average of each branch

- `HAVING` clause
  - Provides a condition to select or reject an entire group
  - If only interested in branches where average balance is > $12000

```
SELECT b-name, AVG(balance)
FROM Deposit
GROUP BY b-name
HAVING AVG(balance) > 12000;
```

# GROUP BY and HAVING Clauses

- **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
        HAVING      COUNT (*) > 2;
```

# EXPANDED Block Structure of SQL Queries

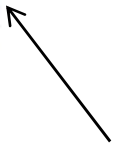
**SELECT** <attribute and function list>
**FROM** <table list>
[ **WHERE** <condition> ]
[ **GROUP BY** <grouping attribute(s)> ]
[ **HAVING** <group condition> ]
[ **ORDER BY** <attribute list> ];

# Views (Virtual Tables)

- `CREATE VIEW` from SELECT
  - In V1, attributes retain the names from base tables. In V2, attributes are assigned new 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;
```



New attribute names

# Views

- Example:

```
CREATE VIEW  loan-info
AS SELECT bname, loan#, cname
FROM Borrow;
```

- Views can be regarded and retrieved as ordinary tables

  E.g.,

```
SELECT loan#, cname
FROM loan-info
WHERE bname = “Central”;
```

# Views

- E.g., suppose we define a view:

```
CREATE VIEW  branch-city
AS SELECT bname, city
FROM Borrow, Customer
WHERE Borrow.cname = Customer.cname;
```

- Now if we need to insert a tuple (Brighton, Shatin) through this view, it will cause:

  (Brighton, null, null, null) => Borrow, and

  (null, null, Shatin) => Customer

  Then what happens to the following query?

```
SELECT * FROM branch-city;
```

  (would the tuple (Brighton, Shatin) be part of the answer?)

# Views

- The answer is NO, because all comparisons involving *null* values are defined to be false!

- To avoid such problems and to simplify implementation, most SQL-based DBMSs restrict the following condition:

> "A modification is permitted through a view ONLY IF the view is defined in terms of ONE base relation."

# NULL Values

- Information can be very often incomplete in the real world
- Unknown attributes are assigned a null value
- One proposal to deal with null values is by using 3-valued logic:

| Not | |
|---|---|
| T | F |
| U | U |
| F | T |

| AND | T | U | F |
|---|---|---|---|
| T | T | U | F |
| U | U | U | F |
| F | F | F | F |

| OR | T | U | F |
|---|---|---|---|
| T | T | T | T |
| U | T | U | U |
| F | T | U | F |

And

- Only T-T returns T
- And-ing F with anything results with F
- The rest is undefined

Or

- Only F-F returns F
- Or-ing T with anything results with T
- The rest is undefined

# NULL Values Comparisons

| Condition | Value of a | Evaluation |
|---|---|---|
| a IS NULL | 10 | FALSE |
| a IS NOT NULL | 10 | TRUE |
| a IS NULL | NULL | TRUE |
| a IS NOT NULL | NULL | FALSE |
| a = NULL | 10 | UNKNOWN |
| a = !NULL | 10 | UNKNOWN |
| a = NULL | NULL | UNKNOWN |
| a = !NULL | NULL | UNKNOWN |
| a = 10 | NULL | UNKNOWN |
| a = 10 | NULL | UNKNOWN |

# NULL Values

- Any problem with the following query?

```
SELECT c-name
FROM Deposit
WHERE balance > 10000
      OR balance <= 10000;
```

(In fact, all comparisons involving Null become FALSE!)

- In most SQL-based DBMSs, the special keyword NULL may be used to test for a null value. E.g.,

```
SELECT c-name
FROM Deposit
WHERE balance IS NULL;
      (or balance IS NOT NULL;)
```

# Data Manipulation Language(DML)

# INSERT Command

- Specify the relation name and a list of values for the tuple

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

```
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 values for the attributes are obtained from the results of the SELECT statement

# DELETE Command

- Removes tuples from a relation

  DELETE FROM table-name;

  - (Note: this operation only deletes all tuples from the table and the table is still there)
  - Includes a `WHERE` clause to select the tuples to be deleted

```
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 Command

- Modify attribute values of one or more selected tuples
- Additional `SET` clause in the `UPDATE` command
  - Specifies attributes to be modified and new values

```
U5:   UPDATE   PROJECT
      SET      Plocation = 'Bellaire', Dnum = 5
      WHERE    Pnumber=10;
```

# UPDATE Command

- *Update*
  - Example1:

    UPDATE Deposit

    SET balance = balance * 1.05

    *(to increase the payment by 5% to all accounts; it is applied to each tuple exactly once.)*
  - Example2:

    UPDATE Deposit

    SET balance = balance * 1.06 WHERE balance > 10000

    UPDATE Deposit

    SET balance = balance * 1.05 WHERE balance <= 10000

    *(to increase the payment by 6% to all accounts with balance over $10000; all others receive 5% increase.)*

# Summary of SQL Syntax

**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> )
```

# Summary of SQL Syntax

**Table 7.2** Summary of SQL Syntax

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

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

# References

- 6e
  - Ch. 4. p. 83 – 107
  - Ch. 5, p.  111 – 126