SQL: Schema Definition, Basic Constraints, and Queries

SQL Introduction

- SQL stands for **S**tructured **Q**uery **L**anguage
- Standard language for querying and manipulating data
- Data Definition Language (DDL)
  - Create/alter/delete tables and their attributes
- Data Manipulation Language (DML)
  - Query one or more tables
  - Insert/delete/modify tuples in tables

Many standards out there: ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), . . .

TABLES IN SQL

Table name

Attribute names

Department

DName	Dnumber	MgrSsn	Mgrstartdate
Gizmo	19	324521	1992-08-11
Powergizmo	29	624545	1982-01-21
SingleTouch	1	986133	1912-04-02
MultiTouch	2	1123455	2002-08-28

Tuples or rows

CREATE TABLE

- Creates a new relation(table) in the database
  - Specifies relation's attributes and their data types
  - Specifies constraints such as NOT NULL , UNIQUE ,CHECK etc…

```
CREATE TABLE   DEPARTMENT
        (DNAME          VARCHAR(10)      NOT NULL  ,
        DNUMBER         INTEGER  CHECK(DNUMBER >0 AND DNUMBER <25),
        MGRSSN          CHAR(9),
        MGRSTARTDATE     DATE
        );
```

CREATE SCHEMA

- Specifies a new database schema by giving it a name

- Example:

  CREATE SCHEMA COMPANY **AUTHORIZATION** Zareen**;**

Data Types

- Numeric
  - integer
  - Smallint
  - Floating point
  - Decimal(Precision, Scale)
  - Numeric(Precision,Scale)
- Character String
  - Char(n)
  - Varchar(n)
  - Nvarchar(n)
- Bit-String
- Bolean

Additional Data Types

- **DATE:**
  - Made up of year-month-day in the format yyyy-mm-dd
- **TIME:**
  - Made up of hour:minute:second in the format hh:mm:ss
- **TIME(i):**
  - Made up of hour:minute:second plus i additional digits specifying fractions of a second
  - format is hh:mm:ss:ii...i
- **TIMESTAMP:**
  - Has both DATE and TIME components

CREATE DOMAIN

- We can declare a domain in SQL
- This makes it easier to change the data type of numerous attributes in a schema
- Improves Schema readability
- Not available in many SQL implementation like (T-SQL)

CREATE DOMAIN SSN_TYPE AS CHAR(9);

CONSTRAINTS IN SQL

- CREATE TABLE command allows us to specify the primary key, secondary keys, and foreign keys.
- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases

```
CREATE TABLE   DEPARTMENT
(       DNAME           VARCHAR(10)     NOT NULL,
        DNUMBER         INTEGER         NOT NULL,
        MGRSSN          CHAR(9)         NULL,
        MGRSTARTDATE    CHAR(9),
        PRIMARY KEY (DNUMBER),
        UNIQUE (DNAME),
        FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE
 );
```

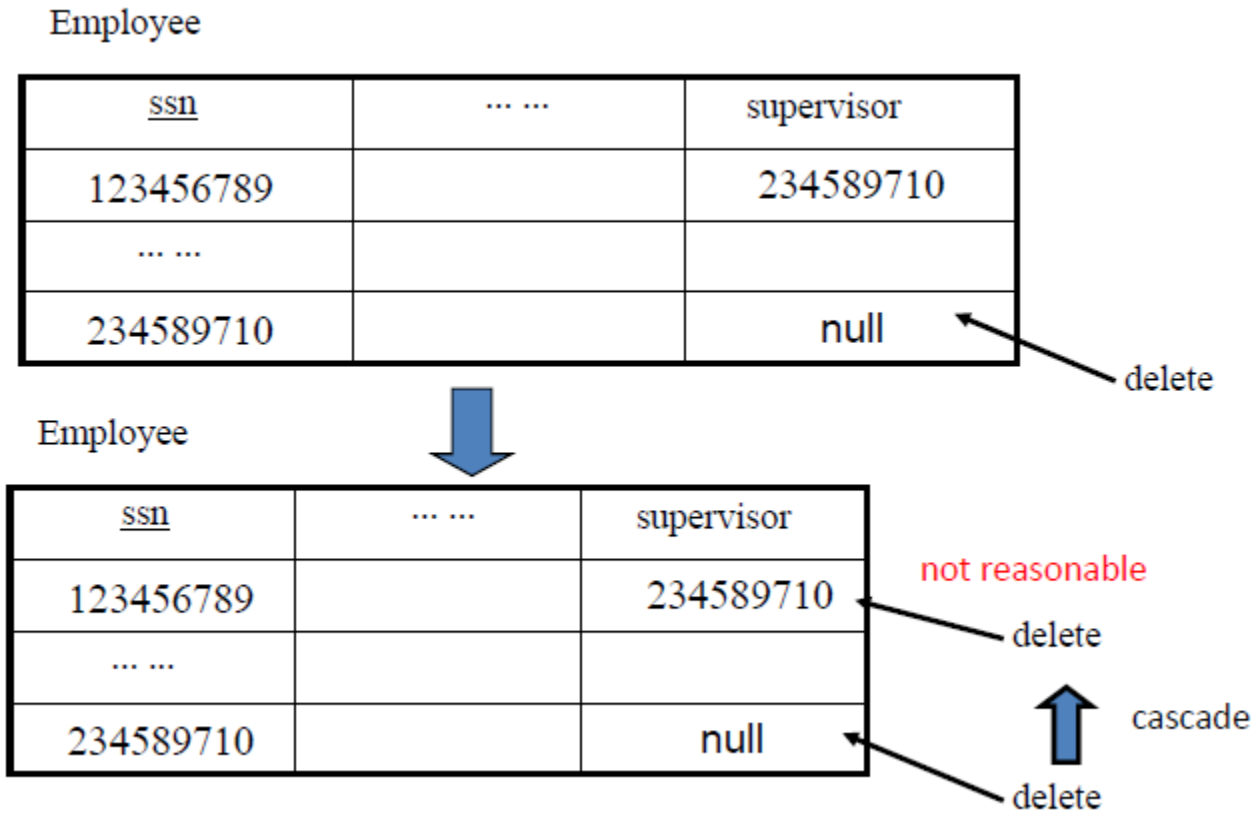
REFERENTIAL INTEGRITY OPTIONS

- We can specify RESTRICT, CASCADE, SET NULL or SET DEFAULT on foreign keys.

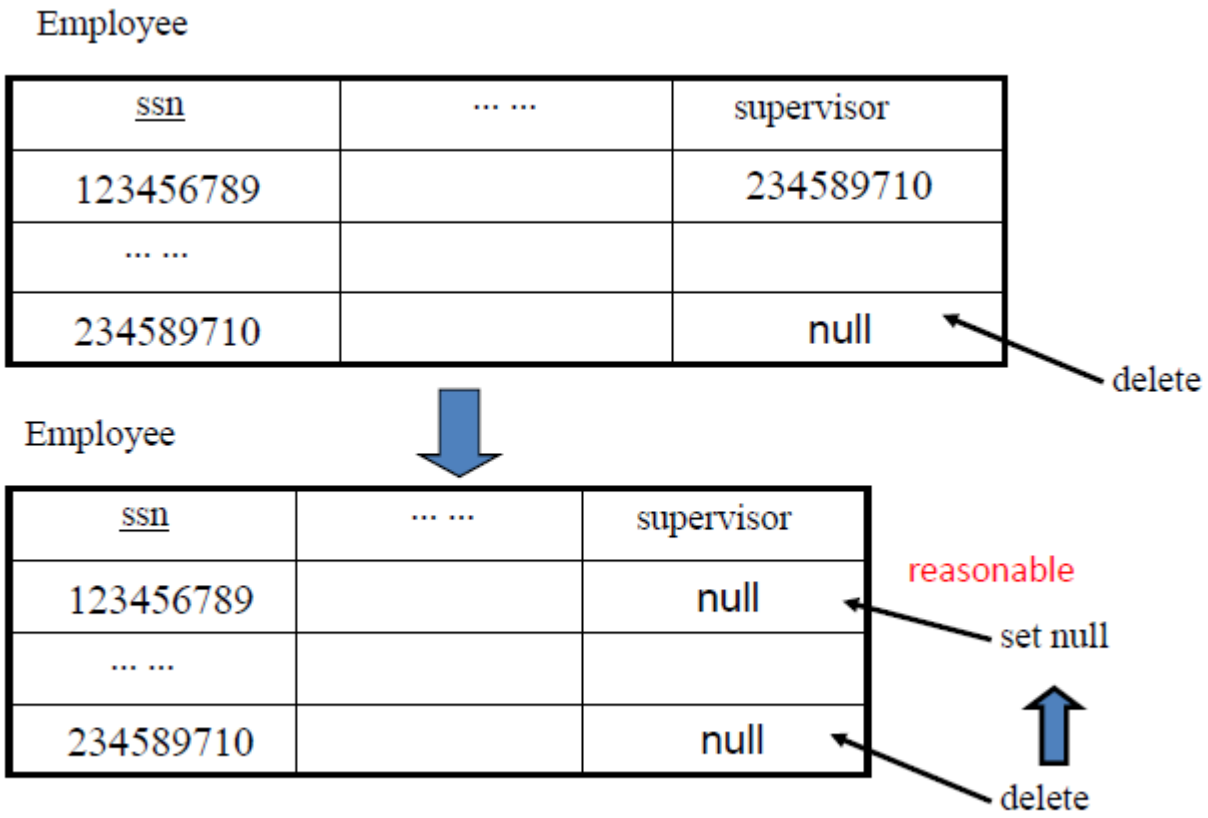
```
CREATE TABLE   DEPARTMENT
 (       DNAME            VARCHAR(10)      NOT NULL,
         DNUMBER          INTEGER          NOT NULL,
         MGRSSN           CHAR(9) ,
         MGRSTARTDATE     CHAR(9),
         PRIMARY KEY (DNUMBER),
         UNIQUE (DNAME),
         FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE
         ON DELETE SET DEFAULT ON UPDATE CASCADE
);
```

CASCADE : to delete a row with a key referenced by foreign keys in existing rows in other tables, all rows that contain those foreign keys are also deleted.

REFERENTIAL INTEGRITY OPTIONS



REFERENTIAL INTEGRITY OPTIONS



REFERENTIAL INTEGRITY OPTIONS

```
CREATE TABLE   EMPLOYEE
        (           ENAME   VARCHAR(30)      NOT NULL,
                    ESSN    CHAR(9),
                    BDATE   DATE,
                    DNO     INTEGER  DEFAULT 1,
                    SUPERSSN          CHAR(9),
                    PRIMARY KEY (ESSN),
                    FOREIGN KEY (DNO) REFERENCES DEPARTMENT
              ON DELETE SET DEFAULT ON UPDATE CASCADE,
                    FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE
              ON DELETE SET NULL ON UPDATE CASCADE
 );
```

SQL CONSTRAINTS

- Assigning Names to Constraints

CONSTRAINT deptPK PRIMARY KEY(Dnumber)
CONSTRAINT deptSK UNIQUE(Dname)

- CHECK Constraint

CHECK (Dept_create_date <= Mgr_start_date)

DROP COMMAND

- Drop Command is used to delete schema or named schema elements such as table, domains, or constraints
- Example:

DROP TABLE DEPENDENT;

DROP TABLE EMPLOYEE CASCADE;

DROP SCHEMA COMPANY;

In SQL-Server (T-SQL), DROP TABLE cannot be used to drop a table that is referenced by a FOREIGN KEY. The referencing FOREIGN KEY or the referencing table must first be dropped.

ALTER COMMAND

- The definition of table or named schema elements can be changed using ALTER command
- ALTER can be used to add an attribute to the relation
  - Initially, the new attribute will have NULLs in all the tuples of the relation
  - NOT NULL constraint is *not allowed* for such an attribute
- **Example :**

  ALTER TABLE EMPLOYEE ADD COLUMN JOB VARCHAR(12);

  T-SQL syntax

  ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);

  - The database user have to enter a value for the new attribute JOB for each EMPLOYEE tuple.

ALTER TABLE

- ALTER command can be use to add or drop constraints
- **Example :**

  ALTER TABLE EMPLOYEE **add constraint** unEmp **UNIQUE(**NAME**)** **;**

  ALTER TABLE EMPLOYEE **drop constraint** unEmp **;**

SQL Queries

Basic form:

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

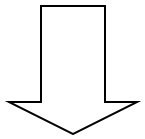
- *Not same as* the SELECT operation of the relational algebra
- The result can have duplicate tuples
- SQL relation is a *multi-set* (bag) of tuples; *not* a set of tuples

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT *
FROM Product
WHERE category='Gadgets'
```



PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

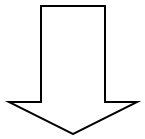
"selection"

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  PName, Price, Manufacturer
FROM    Product
WHERE   Price > 100
```

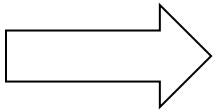


"selection" and "projection"

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

Eliminating Duplicates

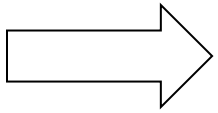
```
SELECT DISTINCT category
FROM Product
```



Category
Gadgets
Photography
Household

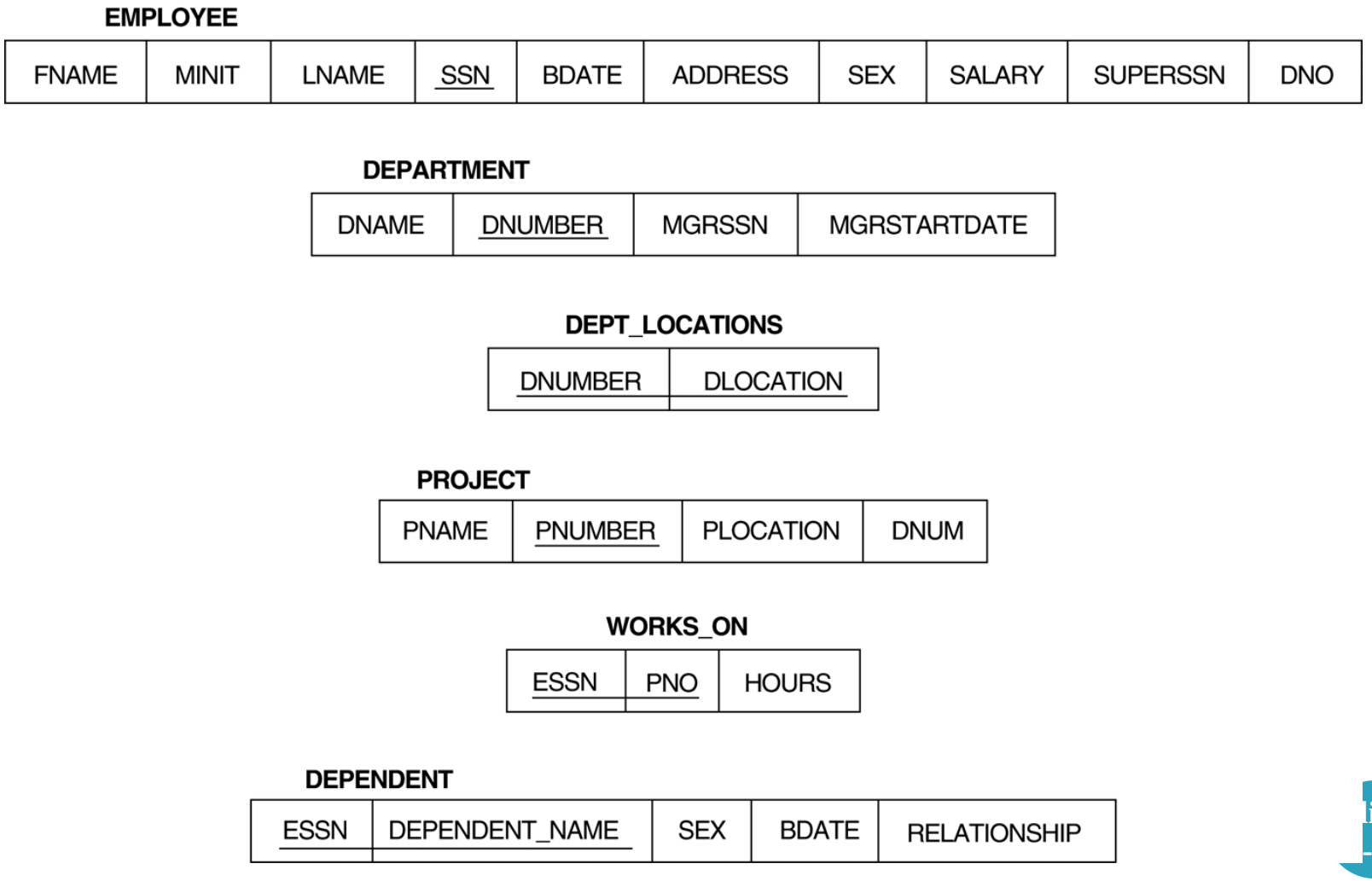
Compare to:

```
SELECT category
FROM Product
```



Category
Gadgets
Gadgets
Photography
Household

Relational Database Schema



Simple SQL Queries

- Basic SQL queries correspond to using the SELECT, PROJECT, and JOIN operations of the relational algebra
- Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

```
SELECT   BDATE, ADDRESS
FROM     EMPLOYEE
WHERE    FNAME='John' AND MINIT='B'
         AND  LNAME='Smith'
```

- Similar to a SELECT-PROJECT pair of relational algebra operations

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

JOIN OPERATION

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

```
SELECT   FNAME, LNAME, ADDRESS
FROM     EMPLOYEE, DEPARTMENT
WHERE    DNAME='Research' AND DNUMBER=DNO
```

- DNAME='Research' is a *selection condition*
- DNUMBER=DNO is a *join condition*

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

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

JOIN(CONT.)

- For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

```
SELECT   PNUMBER, DNUM, LNAME, BDATE, ADDRESS
FROM     PROJECT, DEPARTMENT, EMPLOYEE
WHERE    DNUM=DNUMBER AND MGRSSN=SSN  AND
         PLOCATION='Stafford'
```

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

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

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

UNSPECIFIED WHERE-CLAUSE

- *Missing WHERE-clause*
  - indicates there is no condition and is same as WHERE TRUE
- Retrieve the SSN values for all employees.

```
SELECT   SSN
FROM     EMPLOYEE
```

- If there is no join condition, then we get *CARTESIAN PRODUCT*

```
SELECT   SSN, DNAME
FROM     EMPLOYEE, DEPARTMENT
```

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

USE OF *

- To retrieve all the attribute values of the selected tuples, a * is used, which stands for *all the attributes*

  Examples:

```
SELECT   *
FROM     EMPLOYEE
WHERE    DNO=5

SELECT   *
FROM     EMPLOYEE, DEPARTMENT
WHERE    DNAME='Research' AND   DNO=DNUMBER
```

Aliases

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*
- A query that refers to two attributes with the same name must *prefix* the relation name to the attribute name

Example:

EMPLOYEE.DNO, DEPARTMENT.DNUMBER

ALIASES

- For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.
  - **SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME**
    FROM EMPLOYEE E S
    WHERE E.SUPERSSN=S.SSN

- Can also use the AS keyword to specify aliases
  - **SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME**
    FROM EMPLOYEE AS E, EMPLOYEE AS S
    WHERE E.SUPERSSN=S.SSN

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

ARITHMETIC OPERATIONS

- Arithmetic operators '+', '-'. '*', and '/') can be applied to numeric values in an SQL query result
- Give all employees who work on the 'ProductX' project a 10% raise.
  - **SELECT FNAME, LNAME, 1.1*SALARY**
    FROM EMPLOYEE, WORKS_ON, PROJECT
    WHERE SSN=ESSN AND PNO=PNUMBER AND
    PNAME='ProductX'

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

ORDER BY

- The **ORDER BY** clause sort the tuples in a query result

- Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name, then first name.

```
SELECT      DNAME, LNAME, FNAME, PNAME
FROM        DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE       DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
ORDER BY    DNAME, LNAME,FNAME
```

- The default order is in ascending order of values
- We can specify the keyword **DESC** if we want a descending order
  - **ORDER BY** Dname **DESC,** Lname **ASC**

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL
- SQL uses **IS** or **IS NOT** to compare NULLs
- Retrieve the names of all employees who do not have supervisors.
-

```
SELECT    FNAME, LNAME
FROM      EMPLOYEE
WHERE     SUPERSSN  IS  NULL
```

- Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

SUBSTRING COMPARISON

- **LIKE** operator is used to compare partial strings
- Two reserved characters are used:
  - '%' (or '*' in some implementations) replaces an arbitrary number of characters, and
  - '_' replaces a single arbitrary character
- Retrieve all employees whose address is in Houston, Texas.
-

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       ADDRESS LIKE '%Houston,TX%'
```

SUBSTRING COMPARISON (CONT.)

- Retrieve all employees who were born during the 1950s.
- Here, '5' must be the third character of the string , so the BDATE value is '__5_____',.

```
SELECT    FNAME, LNAME
FROM      EMPLOYEE
WHERE     BDATE LIKE    '__5_____',.
```

- LIKE operator allows us to get around the fact that each value is considered atomic and indivisible;
  - hence, in SQL, character string attribute values are not atomic

JOINED RELATIONS IN SQL

- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)
- **Example:**
  - SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM EMPLOYEE AS E , EMPLOYEE AS S
    WHERE E.SUPERSSN=S.SSN
  - SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM (EMPLOYEE E **JOIN** EMPLOYEE S
    ON E.SUPERSSN=S.SSN
  - SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM (EMPLOYEE E **LEFT OUTER JOIN** EMPLOYEE S
    ON E.SUPERSSN=S.SSN)

Joined Relations Feature in SQL

```
SELECT   FNAME, LNAME, ADDRESS
FROM     EMPLOYEE, DEPARTMENT
WHERE    DNAME='Research' AND DNUMBER=DNO
```

could be written as:

```
SELECT   FNAME, LNAME, ADDRESS
FROM     (EMPLOYEE JOIN DEPARTMENT
          ON DNUMBER=DNO)
WHERE    DNAME='Research'
```

or as:

```
SELECT   FNAME, LNAME, ADDRESS
FROM     (EMPLOYEE NATURAL JOIN DEPARTMENT
          AS DEPT(DNAME, DNO, MSSN, MSDATE)
WHERE    DNAME='Research'
```

JOINED RELATIONS FEATURE IN SQL

- Example that illustrates multiple joins

```
SELECT    PNUMBER, DNUM, LNAME,
FROM      (PROJECT JOIN DEPARTMENT
           ON  DNUM=DNUMBER)
          JOIN EMPLOYEE ON MGRSSN=SSN) )
WHERE     PLOCATION='Stafford'
```

SET OPERATIONS

- SQL has incorporated some set operations like
  - Union operation (**UNION)**,
  - Set difference (**EXCEPT)** and
  - Intersection operation (**INTERSECT)**
- *Duplicate tuples are eliminated from the result*
- Requires *union compatible relations*

SET OPERATIONS (CONT.)

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

```
(SELECT  PNAME
 FROM    PROJECT, DEPARTMENT, EMPLOYEE
 WHERE   DNUM=DNUMBER AND MGRSSN=SSN AND
                      LNAME='Smith')
 UNION
 (SELECT  PNAME
 FROM    PROJECT, WORKS_ON, EMPLOYEE
 WHERE   PNUMBER=PNO AND ESSN=SSN AND
                      LNAME='Smith')
```

NESTING OF QUERIES

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*
- Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT  FNAME, LNAME, ADDRESS
FROM    EMPLOYEE
WHERE   DNO IN  (SELECT  DNUMBER
                 FROM    DEPARTMENT
                 WHERE   DNAME='Research' )
```

	FNAME	LNAME	ADDRESS
1	John	Smith	731 Fondren, Housten, Tx
2	Franklin	Wong	639 Voss, Houston, TX
3	Joyce	English	5631 Rice, Houston, TX
4	Ramesh	Narayan	975 Fire Oak, Humble, TX

	DNUMBER
1	5

CORRELATED NESTED QUERIES

- If a condition in the *nested query* references an attribute of a relation declared in the *outer query* , then two queries are said to be *correlated*
- Retrieve the name of each employee who has a dependent with the same first name and gender as the employee.

```
SELECT  E.FNAME, E.LNAME
FROM    EMPLOYEE AS E
WHERE  E.SSN IN (SELECT ESSN   FROM    DEPENDENT AS D
                 WHERE E.SEX = D.SEX AND FNAME=DEPENDENT_NAME)
```

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

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

Nested query is evaluated once for each tuple in outer query

CORRELATED NESTED QUERIES (CONT.)

- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query.
- For example, the query on previous slide can be written as

Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT    E.FNAME, E.LNAME
FROM      EMPLOYEE E, DEPENDENT D
WHERE     E.SSN=D.ESSN AND
          E.FNAME=D.DEPENDENT_NAME
```

NESTED QUERIES

SQL allows use of **tuples** of values in comparisons

- Select the Essns of all employees who work the same (project, hours) combination that 'John Smith' (whose Ssn = '123456789') works on.

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

EXISTS FUNCTION

- EXISTS Function checks whether the result of a nested query is empty or not
- Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT    FNAME, LNAME
FROM      EMPLOYEE
WHERE     EXISTS  (SELECT *
                   FROM      DEPENDENT
                   WHERE     SSN=ESSN AND
                    FNAME=DEPENDENT_NAME)
```

EXISTS FUNCTION (CONT.)

- Retrieve the names of employees who have no dependents.

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       NOT EXISTS   (SELECT  *
                          FROM  DEPENDENT
                          WHERE SSN=ESSN)
```

  - The above correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple.
    - If *none exist* , the EMPLOYEE tuple is selected
  - EXISTS is necessary for the expressive power of SQL

EXISTS FUNCTION (CONT.)

- Find the names of managers who have at least one dependents.

```
SELECT   FNAME, LNAME
FROM              EMPLOYEE
WHERE  EXISTS   (SELECT   *
                    FROM  DEPENDENT
                    WHERE SSN=ESSN)
          AND
          EXISTS   (SELECT   *
                    FROM  DEPARTMENT
                    WHERE SSN=Mgr_SSN)
```

EXISTS FUNCTION (CONT.)

- Retrieve the name of each employee who works on *all* the projects controlled by department number 5.
- Set theory: S1 contains S2 if (S2 – S1 = 0)

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    NOT EXISTS (
            (SELECT  PNUMBER
             FROM PROJECT
             WHERE     DNUM=5)
             EXCEPT
            (SELECT      PNO
             FROM        WORKS_ON
             WHERE       SSN=ESSN)
)
```

S1 = set of projects of each employee
S2= set of DN0=5 projects

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

NESTED QUERIES

CONTAINS compares two *sets* , and returns TRUE if one set contains all values in the other set.

Same as *division* operation of relational algebra

- Retrieve the name of each employee who works on *all* the projects controlled by department number 5.
-

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE  ( (SELECT    PNO
          FROM      WORKS_ON
          WHERE     SSN=ESSN)
          CONTAINS
         (SELECT    PNUMBER
          FROM      PROJECT
          WHERE     DNUM=5) )
```


Most implementations of SQL *do not* have this operator

NESTED CORRELATED QUERIES (CONTD)

You can also use: s > ALL R
s > ANY R
EXISTS R

Product (pname, price, category, maker)
Find products that are more expensive than all those produced By “IBM”

```
SELECT  name
FROM    Product
WHERE  price >  ALL (SELECT price
                     FROM    Purchase
                     WHERE  maker=‘IBM’)
```

NESTED CORRELATED QUERIES (CONTD)

You can also use: s > ALL R
s > ANY R
EXISTS R

Find Employee whose salary is greater than the salary of all employee in department 5

```
SELECT  Fname
FROM    Employee
WHERE  Salary >  ALL (SELECT Salary
                      FROM    Employee
                      where Dno=5)
```

	Fname
1	James
2	Jennifer

Complex Correlated Query

Product (pname, price, category, maker, year)

- Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT DISTINCT  pname, maker
FROM     Product AS x
WHERE  price > ALL  (SELECT  price
                          FROM    Product AS y
                          WHERE  x.maker = y.maker AND x.pname!=y.pname
                                   and
                                   y.year < 1972)
```

Very powerful ! Also much harder to optimize.

Complex Correlated Query

- Find Employee (his dno and salary) whose salary is greater than all employees in the same department

```
SELECT   Fname, Salary, Dno
 FROM    Employee as E
 WHERE  Salary >  ALL (SELECT Salary
                 FROM    Employee as S
                 WHERE   E.dno=S.dno and E.ssn !=S.ssn )
```

	Fname	salary	Dno
1	Franklin	40000	5
2	James	55000	1
3	Jennifer	43000	4

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
SELECT    DISTINCT  ESSN
FROM      WORKS_ON
WHERE     PNO IN  (1, 2, 3)
```

AGGREGATE FUNCTIONS

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Find the maximum salary, the minimum salary, and the average salary among all employees.

```
SELECT    MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM      EMPLOYEE
```

  - Some SQL implementations *may not allow more than one function* in the SELECT-clause

AGGREGATE FUNCTIONS (CONT.)

- Retrieve the number of employees in the 'Research' department

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

GROUPING

- **GROUP BY**-clause specifies the grouping attributes
- For each department, retrieve the department number, the number of employees in the department, and their average salary.
- **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	Borg	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

GROUPING (CONT.)

- For each project, retrieve the project number, project name, and the number of employees who work on that project.
-

```
SELECT   PNUMBER, PNAME, COUNT (*)
FROM     PROJECT, WORKS_ON
WHERE    PNUMBER=PNO
GROUP BY PNUMBER, PNAME
```

	PNUMBER	PNAME	count
1	1	ProductX	2
2	2	ProductY	3
3	3	ProductZ	2
4	10	Compu...	3
5	20	Reorga...	3
6	30	Newbe...	3

  - The grouping and functions are applied *after* the joining of the two relations
  - Group By clause specifies grouping attributes which should appear in SELECT clause

HAVING-CLAUSE

- HAVING-clause specify a selection condition on groups
- For each project *on which more than two employees work* , retrieve the project number, project name, and the number of employees who work on that project.
-

```
SELECT      PNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
HAVING      COUNT (*) > 2
```

HAVING-CLAUSE

(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

PNUMBER	PNAME	count
2	ProductY	3
10	Computerization	3
20	Reorganization	3
30	Newbenefits	3

HAVING-CLAUSE

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)

Group By and Having

- Count the *total* number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work

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

General form of Grouping and Aggregation

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

Evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes $a_1,\ldots,a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Two Examples

Store(sid, sname)
Product(pid, pname, price, sid)

Find stores that sell *only* products with price > 100

same as:

Find stores s.t. all their products have price > 100)

```
SELECT Store.name
FROM   Store, Product
WHERE  Store.sid = Product.sid
GROUP BY  Store.sid, Store.name
HAVING 100 < min(Product.price)
```

Find stores s.t. all their products have price > 100

Almost equivalent…

```
SELECT Store.name
FROM   Store
WHERE
   100 < ALL (SELECT Product.price
              FROM product
              WHERE Store.sid = Product.sid)
```

```
SELECT Store.name
FROM   Store
WHERE  Store.sid NOT IN
          (SELECT Product.sid
           FROM Product
           WHERE  Product.price <= 100)
```

Aggregate Example

- **Example:** Count the number of distinct salary values in the database.
  - **SELECT COUNT** (**DISTINCT** Salary)
  - **FROM** EMPLOYEE;
- NULL values are **discarded** when aggregate functions are applied to a particular attribute.

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

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

- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query

SQL Queries

- There are various ways to specify the same query in SQL
  - This is to give flexibility to user to specify queries
- For query optimization, it is preferable to write a query with as little nesting and implied ordering as possible.
- Ideally, DBMS should process the same query in the same way regardless of how the query is specified.
  - But this is quite difficult in practice, (chapter 19,20)

Specifying Updates in SQL

- There are three SQL commands to modify the database;
  - INSERT,
  - DELETE, and
  - UPDATE

INSERT

- It is used to add one or more tuples to a relation

- **Example:**

```
INSERT INTO  EMPLOYEE
VALUES ('Richard','K','Marini', '653298653', '30-DEC-52',
'98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4 )
```

- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

INSERT (CONT.)

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple

- **Example:** Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
        VALUES ('Richard', 'Marini', '653298653')
```

- Attributes with NULL values can be left out

INSERT (CONT.)

- Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department.
- A table DEPTS_INFO is created by Q1, and is loaded with the information retrieved from the database by the query Q2.

```
Q1:     CREATE TABLE  DEPTS_INFO
                (DEPT_NAME      VARCHAR(10),
                 NO_OF_EMPS     INTEGER,
                 TOTAL_SAL      INTEGER);

Q2:     INSERT INTO     DEPTS_INFO (DEPT_NAME,
                        NO_OF_EMPS, TOTAL_SAL)
        SELECT          DNAME, COUNT (*), SUM (SALARY)
        FROM            DEPARTMENT, EMPLOYEE
        WHERE           DNUMBER=DNO
        GROUP BY        DNAME ;
```

DELETE

- Removes tuples from a relation
- Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
- Examples:

```
DELETE FROM    EMPLOYEE
WHERE          LNAME='Brown'


DELETE FROM    EMPLOYEE
WHERE          DNO      IN (SELECT  DNUMBER
               FROM    DEPARTMENT
               WHERE DNAME='Research')


DELETE FROM    EMPLOYEE
```

UPDATE

- Used to modify attribute values of selected tuples
- **Example:** Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

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

UPDATE (CONT.)

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

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

Views in SQL

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations (since the table may not physically be stored)
- Allows full query operations
- A convenience for expressing certain operations
- They are used to:
  - simplify complex queries, and
  - define distinct conceptual interfaces for different users.

SQL Views: An Example

- Specify a different WORKS_ON table

```
CREATE VIEW WORKS_ON1 AS
        SELECT FNAME, LNAME, PNAME, HOURS
        FROM EMPLOYEE, PROJECT, WORKS_ON
        WHERE SSN=ESSN AND PNO=PNUMBER
```

WORKS_ON1

Fname	Lname	Pname	Hours

SQL VIEWS: AN EXAMPLE2

```
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;
```

DEPT_INFO

Dept_name	No_of_emps	Total_sal

Using a Virtual Table

- We can specify SQL queries on a newly created view:

```
SELECT FNAME, LNAME
FROM WORKS_ON1
WHERE PNAME='ProductX';
```

- DBMS is responsible to keep view always up-to-date
- When no longer needed, a view can be dropped:

```
DROP WORKS_ON1;
```

Efficient View Implementation

- **Query modification:** present the view query in terms of a query on the underlying base tables

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

```
SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSN=ESSN AND PNO=PNUMBER AND PNAME='PRODUCTX'
```

- Disadvantage:
- Inefficient for views defined via complex queries (esp if additional queries are to be applied within a short time period)

Efficient View Implementation

- **View materialization**: involves physically creating and keeping a temporary table
  - assumption: other queries on the view will follow
  - concerns: maintaining correspondence between the base table and the view when the base table is updated
  - strategy: incremental update

View Update

- **Single view without aggregate operations**:
  - update may map to an update on the underlying base table
- **Views involving joins**:
  - an update *may* map to an update on the underlying base relations
  - not always possible
- **Example:**

```
UPDATE WORKS_ON1
SET PNAME=XYZ'
WHERE FNAME='JOHN AND
      LNAME='SMITH' AND
        PNAME='PRODUCTX'
```

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

A) UPDATE WORKS_ON1

SET PNO = (SELECT PNUMBER FROM PROJECT

WHERE PNAME='XYZ')

WHERE ESSN IN (SELECT SSN FROM EMPLOYEE

WHERE LNAME='SMITH' AND FNAME='JOHN')

AND

PNO = (SELECT PNUMBER FROM PROJECT

WHERE PNAME='PRODUCTX')

B)UPDATE PROJECT SET PNAME='XYZ'

WHERE PNAME='PRODUCTX'

Un-updatable Views

- Views defined using groups and aggregate functions are not updateable

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

- Views defined on multiple tables using joins are generally not updateable
- `WITH CHECK OPTION`: must be added to the definition of a view if the view is to be updated
  - to allow check for updatability and to plan for an execution strategy

ASSERTION

- In SQL, table constraints are associated with a single table.
  - **CHECK(DNUMBER >0 AND DNUMBER <25)**
  - **CHECK (Dept_create_date <= Mgr_start_date)**

- Expression in the **CHECK** clause can refer to other tables but when a constraint involves many tables, it becomes cumbersome

- Assertions are constraints that are not associated with any one table.

- SQL-server do not support assertion, however it do support trigger and check constraint

Assertions: An Example

- “The salary of an employee must not be greater than the salary of the manager of the department that the employee works for’’

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT *
                   FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
                   WHERE E.SALARY > M.SALARY AND
                   E.DNO=D.NUMBER AND D.MGRSSN=M.SSN))
```

Using General Assertions

- Specify a query that violates the condition; include inside a `NOT EXISTS` clause
- Query result must be empty
  - if the query result is not empty, the assertion has been violated

- **Create Assertion to enforce the** constraint that
  - The number of Employees in each department should be less than 20.
  - An employee should be assign a project of his\her department only.
  - The total working hours of all the employees in a department should be less than 180.

- CHECK clause can also be used to specify constraints on *individual* attributes, domains and *individual* tuples

CONSTRAINT deptPK PRIMARY KEY(Dnumber)

CONSTRAINT deptSK UNIQUE(Dname)

CHECK (Dept_create_date <= Mgr_start_date)

- Difference between CREATE ASSERTION and the individual domain and tuple constraints is
  - CHECK on individual attributes, domains, and tuples are checked *only when tuples are inserted or updated*.

SQL Triggers

- Objective: to monitor a database and take action when a condition occurs

- Triggers are expressed in a syntax similar to assertions and include the following:
  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)

SQL Triggers: An Example

- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
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;
```

Example: Boat Rental database

- Consider the following **Boat Rental database** schema
  - **SAILOR (SID, SName, Phone, City)**
  - **BOAT (BName, BType, Price, OID)**
  - **RESERVATION (SID, BName, Date, Duration)**
  - **OWNER (OID, OName, Phone, Street, City, Country)**

- Find the name and city of sailors who reserve every 'Fishing' boat.
- Find the names of boats that are reserved by at least ten different sailors.
- List names, owner names, and prices of the boats which were reserved in 2007 but not in 2008.

EXAMPLE: BOAT RENTAL DATABASE

- Consider the following schema
  - SAILOR (SID, SName, Phone, City)
  - BOAT (BName, BType, Price, OID)
  - RESERVATION (SID, BName, Date, Duration)
  - OWNER (OID, OName, Phone, Street, City, Country)

- **Find the name and city of sailors who reserve every 'Fishing' boat.**
- **SELECT SNAME, CITY**
  FROM SAILOR s
  WHERE NOT EXISTS ((SELECT BNAME
  FROM BOAT
  WHERE BTYPE='Fishing')
  EXCEPT
  (SELECT BNAME
  FROM reservation r
  WHERE r.SID=s.SID))

Example: Boat Rental database

- Consider the following schema
  - SAILOR (SID, SName, Phone, City)
  - BOAT (BName, BType, Price, OID)
  - RESERVATION (SID, BName, Date, Duration)
  - OWNER (OID, OName, Phone, Street, City, Country)

- **Find the names of boats that are reserved by at least ten different sailors.**
-
```
Select bname
From reservation r
Group by bname
Having count(DISTINCT SID) >9
```

Example: Boat Rental database

- Consider the following schema
  - SAILOR (SID, SName, Phone, City)
  - BOAT (BName, BType, Price, OID)
  - RESERVATION (SID, BName, Date, Duration)
  - OWNER (OID, OName, Phone, Street, City, Country)

- **List names, owner names, and prices of the boats which were reserved in 2007 but not in 2008.**
- **Select** distinct b.bname, b.price, o.oname

  From reservation r, boat b, owner o

  Where r.bname = b.bname and b.oid=o.oid and r.date LIKE '%2007%' and r.sid not in

  (Select ic.sid

  From reservation ic, boat ip

  Where ic.bname = ip.bname and ip.date LIKE '%2008%')

Example: Boat Rental database

- **Consider the following Boat Rental database schema:**
  - SAILOR (SID, SName, Phone, City)
  - BOAT (BName, BType, Price, OID)
  - RESERVATION (SID, BName, Date, Duration)
  - OWNER (OID, OName, Phone, Street, City, Country)

- SELECT DISTINCT Bname
- FROM BOAT
- WHERE Price > ALL (SELECT price
  FROM BOAT b , OWNER o
  WHERE b.oid=o.oid and
  Country='Pakistan')

What does the query do?

Example: Boat Rental database

- **Consider the following Boat Rental database schema:**
  - SAILOR (SID, SName, Phone, City)
  - BOAT (BName, BType, Price, OID)
  - RESERVATION (SID, BName, Date, Duration)
  - OWNER (OID, OName, Phone, Street, City, Country)

- Select bname,count(*)
- From reservation r ,boat b,owner o
- Where b.bname=r.bname and b. oid=o.oid and country='Pakistan'
- Group by bname
- Having count(*) > 5

What does the query do?