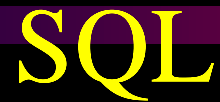
CS5300
Database Systems



SQL

A.R. Hurson

323 CS Building

hurson@mst.edu

Database Systems

Note, this unit will be covered in five lectures. In case you finish it earlier, then you have the following options:

1) Take the early test and start CS5300.module4
2) Study the supplement module (supplement CS5300.module3)
3) Act as a helper to help other students in studying CS5300.module3

Note, options 2 and 3 have extra credits as noted in course outline.

Database Systems

Enforcement of background

Glossary of prerequisite topics

Familiar with the topics?

No → Review CS5300.module3.background

Yes

Take Test

Pass?

No → Remedial action

Yes

At the end: take exam, record the score, impose remedial action if not successful

Current Module

Glossary of topics

Familiar with the topics?

No → Take the Module

Yes

Take Test

Pass?

No

Yes

Options

Study next module?

Lead a group of students in this module (extra credits)?

Study more advanced related topics (extra credits)?

Extra Curricular activities

- You are expected to be familiar with:
  - Basic principles of relational database model,
  - Basic format of SQL
    - Data Definition Language
    - Data Manipulation Language
- If not, you need to study CS5300.module3.background

Database Systems

- Structured Query Language (SQL)
  - SQL is a comprehensive language and provides statements for Data definition and Data manipulation. Hence, it is both a Data Definition Language (DDL) and Data Manipulation Language (DML)

Database Systems

- Structured Query Language (SQL)
  - The SQL language has the following features:
    - Embedded and Dynamic facilities to allow SQL code to be called from a host language or a query be constructed at run-time.
    - Triggers which are actions executed by DBMS whenever certain changes to the database meet certain conditions.

Database Systems

- Structured Query Language (SQL)
  - Security to control users' accesses to data objects.
  - Transaction management commands to allow the execution of transactions.
  - Remote database accesses to allow client server or distributed environments.

- Data Definition Language
  - CREATE SCHEMA
  - CREATE TABLE
  - CREATE DOMAIN
  - CREATE VIEW
  - DROP TABLE
  - DROP VIEW
  - INSERT
  - UPDATE
  - DELETE
  - ALTER
  - Define KEY CONSTRIANTS
  - CHECK

Database Systems

- Running Example

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_Location

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Database Systems

◆Data Definition Language

- CREATE SCHEMA: The general format is as follows:

  CREATE SCHEMA schema-name AUTHORIZATION 'name'

  schema name   authorization identifier

- Schema is identified by a name, and includes an authorization indicating the owner and descriptors for each element in the schema.
- Schema elements are: tables, constraints, views, domains, …

- Running Example

CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith'

Database Systems

◆Data Definition Language

- **CREATE TABLE**: The general format is as follows:

  CREATE TABLE base-table

  (column-definition [, column-definition] …

  [, primary-key-definition]

  [, foreign-key-definition [, foreign-key-definition] …]);

  where a "column-definition" is in the form of

  column data-type [NOT NULL]

- Note that the specification of primary-key is optional.

CREATE TABLE S

```
( S#        CHAR(5)    NOT NULL,
  SNAME     CHAR(20)   NOT NULL,
  STATUS    INTEGER    NOT NULL,
  CITY      CHAR(15)   NOT NULL,
  PRIMARY KEY ( S# ) ) ;
```

- This will create an empty table. The data values now can be entered using INSERT command.

Result:

S	S#	SNAME	STATUS	CITY

Database Systems

- Running Example

CREATE TABLE COMPANY.EMPLOYEE

Or

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 ),
  FOREIGN KEY (Super_ssn)  REFERENCES EMPLOYEE (Ssn),
  FOREIGN KEY (Dn0)  REFERENCES DEPARTMENT (Dnumber) ) ;
```

- Data Definition Language
  - It is possible to specify the data type of each attribute directly, or one can define a domain.

CREATE DOMAIN domain_name AS CHAR(n);

- Data Definition Language
  - It is possible to define a default value for an attribute by appending the clause

    DEFAULT <value> to an attribute definition.
  - The default value is included in any new tuple if an explicit value is not provided for that attribute.

Database Systems

- Running Example

```
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     DEFAULT   1,
          PRIMARY KEY ( Ssn ),
          FOREIGN KEY (Super_ssn)  REFERENCES EMPLOYEE (Ssn),
          FOREIGN KEY (Dn0)  REFERENCES DEPARTMENT (Dnumber) ) ;
```

- Data Definition Language
  - CHECK clause (The keyword CHECK along with a conditional expression) can be used to restrict (enforce a constraint over) attribute or domain value.

    attribute name domain CHECK (conditional expression on attribute name)

Database Systems

- Constraints over a single table

```
CREATE TABLE Students
    ( Sid    CHAR(20),
      name   CHAR(30),
      login  CHAR(20),
      age    INTEGER,
      gpa    REAL ,
      UNIQUE (name, age),
      CONSTRAINT Studentskey
      PRIMARY KEY (Sid)
      CHECK (age >= 16 AND age <=30))
```

- When a new tuple is inserted into the table or an existing tuple is modified, the conditional statement is checked. If the result is false, the command is rejected.

Running Example

Dnumber INT NOT NULL CHECK (Dnumber > 0 and Dnumber < 21);

CREATE DOMAIN D_NUM AS INTEGER CHECK (D_NUM >0 AND D_NUM <21);

- Integrity Constraints
  - As noted before, constraints can be defined either as the table constraints (over a single table) or assertions.

Database Systems

- Constraints over a single table
  - Primary Key/Candidate Key

```
CREATE TABLE Student ( Sid    CHAR(20),
                       name   CHAR(30),
                       login  CHAR(20),
                       age    INTEGER,
                       gpa    REAL ,
                       UNIQUE (name, age),
                       CONSTRAINT Studentskey
                        PRIMARY KEY (Sid) )
```

Studentskey is called the constraint name - It will be returned if the constraint is violated.

Database Systems

- Constraints over a single table
  - Foreign Key
    - Key words FOREIGN KEY and REFERENCE are used to specify this constraint:

```
CREATE TABLE Enroll ( Sid      CHAR(20),
                      Cid      CHAR(20),
                      grade    CHAR(10),
                      PRIMARY KEY (Sid, Cid),
                      FOREIGN KEY (Sid)
                      REFERENCES Students)
```

Referenced relation

Database Systems

- Last lecture
  - Evolution of database computation platform
    - Centralized
    - Client/server
    - Peer to peer
    - Distributed
    - ???
  - SQL
    - Data Definition Language

- Integrity Constraints
  - As we discussed in introduction section, a referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key is modified. The default action taken by SQL is to reject the update operation that causes a violation. Alternatively, one can use a referential triggered action clause to any foreign key.
  - SET NULL, CASCADE, SET DEFAULT on ON DELETE or ON UPDATE actions will do the job.

Database Systems

- Running Example

```
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    DEFAULT  1,
      PRIMARY KEY ( Ssn ),
      FOREIGN KEY (Super_ssn)  REFERENCES EMPLOYEE (Ssn)
                  ON DELETE SET NULL    ON UPDATE   CASCADE,
      FOREIGN KEY (Dn0)  REFERENCES DEPARTMENT (Dnumber)
                  ON DELETE SET DEFAULT    ON UPDATE   CASCADE ) ;
```

If tuple for supervising employee is deleted then Super_ssn for all employee referencing this person will be set to NULL

If Ssn for supervising employee is updated then Super_ssn for all employee referencing this person will be updated

Database Systems

- Data Definition Language
  - INSERT command can be used to insert a single tuple to a relation.

```
INSERT
INTO      S ( S#, SNAME, STATUS, CITY)
VALUES  ('S1', 'SMITH', 10, 'LONDON' );
```

  - or

```
INSERT  INTO      S
VALUES  ('S1', 'SMITH', 10, 'LONDON' );
```

RESULT

S#	SNAME	STATUS	CITY
S1	SMITH	10	LONDON

- Running Example

INSERT INTO EMPLOYEE

VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98 Oak Forest, Katy, TX', 'M', 37000, '987654321', 4);

INSERT INTO EMPLOYEE (FNAME, LNAME, DNO, SSN)

VALUES ('Richard', 'Marini', 4, '653298653');

- Data Definition Language
  - It is possible to insert multiple tuples into a relation by a single INSERT command. In this case, the attribute values forming a tuple are enclosed in parentheses separated by commas.

- Data Definition Language
  - ALTER: Allows to change the definition of a base-table or any named schema element (i.e., add (drop) a new attribute (column) to (from) an existing base-table, add (drop) table constraints, changing a column definition).
  - Its general format is as follows:

- Data Definition Language

  ALTER TABLE base-table ADD column data-type

  ALTER TABLE base-table DROP column CASCADE (RESTRICT)

  - To drop a column we must use either CASCADE or RESTRICT. In case of CASCADE, all constraints and views that reference the column are dropped. In case of RESTRICT, the command is successful if the column is not referenced by other entities in the schema.

Database Systems

ALTER TABLE S ADD DISCOUNT INTEGER;

RESULT

S	S#	SNAME	STATUS	CITY	DISCOUNT

- Discount column is added (at the right) to the table *S*. All existing tuples are (conceptually) expanded, and the value of the new column is null in every record unless a default value is defined).
- Update command is used to define values for "DISCOUNT" in every tuples in S.
- Specification of NOT NULL is not allowed in ALTER TABLE).

Running Example

ALTER TABLE COMPANY.EMPLOYEE ADD JOB VARCHAR(12);

ALTER TABLE COMPANY.EMPLOYEE DROP ADDRESS CASCADE;

- It is also possible to ALTER column definition:

ALTER TABLE COMPANY.DEPARTMENT ALTER MGRSSN DROP DEFAULT;

ALTER TABLE COMPANY.DEPARTMENT ALTER MGRSSN SET DEFAULT “123456789”;

Database Systems

- Data Definition Language
  - DROP TABLE: Allows to destroy an existing base-table.

```
DROP TABLE    base-table;
```

- Example
  - As discussed before command CREATE can be used to define a domain:

```
CREATE DOMAIN  Qtyvalue   INTEGER   DEFAULT 1
                     CHECK (VALUE >= 1 AND VALUE <=1000)
```

  - Here INTEGER is the base type for domain Qtyvalue, however its contents is restricted by the CHECK statement.

- Example
  - Once a domain is defined, it can be used to limit contents of a column in a table.
  - The DEFAULT keyword assigns a default value to a domain — this value will be automatically assumed in an attempt to insert a tuple into the relation without an initial value for an attribute defined over Qtyvalue.

Database Systems

- Constraints over a single table
  - Assume the following tables:
    - Sailors(sid:integer, sname:string, rating:integer, age:real)
    - Boats(bid:integer, bname:string, color:string)
    - Reserves(sid:integer, bid:integer, day:date)
  - Define a constraint that “Interlake” boat cannot be reserved.

Database Systems

- Constraints over a single table

CREATE TABLE Reserves

```
( Sid       INTEGER,
  bid       INTEGER,
  day     DATE,
  PRIMARY KEY (Sid, bid),
  FOREIGN KEY (Sid) REFERENCES Sailors)
  FOREIGN KEY (bid) REFERENCES Boats)
  CONSTRAINT   noInterlakeRes.
  CHECK (“Interlake” <>
                    (SELECT    B.bname
                     FROM      Boats  B
                     WHERE     B.bid = Reserve.bid)))
```

Database Systems

- Data Manipulation Language (DML)
  - SQL provides four DML statements:
    - SELECT,
    - UPDATE,
    - DELETE, and
    - INSERT.

Database Systems

- Data Manipulation Language (DML)

SELECT $A_1, A_2, \ldots, A_n$
FROM $r_1, r_2, \ldots, r_m$
WHERE P

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

SELECT specifies field (s) FROM a specific table (s) WHERE specific condition (s) is true.

Database Systems

SELECT	[DISTINCT] item(s)
FROM	table (s)
[WHERE	condition]
[GROUP BY	field (s)]
[ORDER BY	field (s)];

Target list, a list of attributes

Relation list

Qualifier — expressions involving constants and/or column names combined using AND, OR, and NOT.

- The select Clause
  - SQL allows duplicates. In cases where we want to eliminate duplicate, we must use the keyword DISTINCT.
  - The select clause allows arithmetic operations involving +. -, *, and / operations on constant or attributes of tuples:

```
SELECT      loan-number, branch-name, amount * 100
FROM        loan
```

- The where Clause
  - SQL uses the logical connectors and, or, and not. The operands of logical connectors can be expressions involving <, <=, >, >+, =, and < >.
  - SQL allows between (not between) comparison operator:

```
SELECT    loan-number
FROM      loan
WHERE     amount between 90000 and 100000
```

Database Systems

- The following strategy is used to evaluate an SQL expression:
  - Compute the cross-product of relation-list,
  - Discard resulting tuples if they fail qualifications (restrict),
  - Delete attributes that are not in target-list (project).
  - If DISTINCT is specified, then duplicate tuples are eliminated.

Database Systems

- The following tables are assumed for the rest of this section:

Supplier Relation
S

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

Database Systems

Part
Relation
P

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

Database Systems

SP
Relation

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

Database Systems

- Simple Queries

```
SELECT    S#, STATUS
FROM      S
WHERE     CITY = 'Paris';
```

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

RESULT

S#	Status
S_2	10
S_3	30

Database Systems

- Simple Retrieval

```
SELECT   P#
FROM     SP;
```

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

Duplicates are not removed

RESULT

P#
P_1
P_2
P_3
P_4
P_5
P_6
P_1
P_2
P_2
P_2
P_4
P_5

Database Systems

SELECT DISTINCT P#
FROM SP;

RESULT

P#
P_1
P_2
P_3
P_4
P_5
P_6

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

- Running Example

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

  - In relational algebra

$$\Pi_{BDATE,\ ADDRESS}(\sigma_{FNAME='John'\ AND\ MINIT='B'\ AND\ LNAME='Smith'}(EMPLOYEE))$$

- Running Example

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

  - In relational algebra

$$\Pi_{FNAME,\ LNAME,\ ADDRESS}(\sigma_{DNAME='Research'\ AND\ DNUMBER=DNO}(EMPLOYEE \times DEPARTMENT))$$

Database Systems

- Retrieval of Computed Values: Assume weight in 'Part relation' is in Pound;

SELECT P.P#, 'Weight in Grams = ', P.Weight * 454

FROM P;

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

Result

P#		
P_1	Weight in Grams =	5448
P_2	Weight in Grams =	7718
P_3	Weight in Grams =	7718
P_4	Weight in Grams =	6356
P_5	Weight in Grams =	5448
P_6	Weight in Grams =	8626

Database Systems

- Naming Fields in the resultant relation
  - AS and = are two ways to name fields in result;

```
SELECT    Supplier name = Sname, STATUS
FROM      S
WHERE     CITY = 'Paris';
```

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

Result

Supplier name	Status
Jones	10
Blake	30

Database Systems

```
SELECT    Sname As Supplier name, STATUS
FROM      S
WHERE     CITY = 'Paris';
```

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

Result

Supplier name	Status
Jones	10
Blake	30

- Running Example
  - What is the meaning of the following query?

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

Database Systems

- LIKE is the keyword that allows string matching (pattern matching) operation;

```
SELECT      Sname As Supplier name, City
FROM        S
WHERE       CITY LIKE '%s' ;
```

- Note that '_' stands for any one character (don't care), and '%' stands for 0 or more arbitrary characters (repeated don't care).

Database Systems

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

Result

Supplier name	City
Jones	Paris
Blake	Paris
Adams	Athens

Database Systems

- Get all parts whose names begin with the letter *C*.

```
SELECT    P.*
FROM      P
WHERE     P.Pname LIKE 'C%';
```

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

Result

P#	Pname	Color	Weight	City
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Database Systems

- Similarly, NOT LIKE can also be used in the WHERE clause;

```
SELECT    P.*
FROM      P
WHERE     P.City NOT LIKE '%E%';
```

In this case, the condition is evaluated to "true" if *City* does not contain an 'E'.

Database Systems

```
SELECT    S#
FROM      S
WHERE     Status > 25;
```

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

Result

S#
S_3
S_5

- Retrieval Involving NULL

  Assume for the sake of the example that supplier S_5 has a status value of null.

  Get supplier numbers for supplier with status greater than 25;

Database Systems

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	$\perp$	Athens

Result

S#
S_3

Unlike previous case S_5 does not qualify.

Database Systems

- Get full detail of all suppliers

```
SELECT    *
FROM      S;
```

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

- This is equivalent to:

```
SELECT  S.S#, S.Sname, S.Status, S.City
FROM    S;
```

RESULT

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

Database Systems

- Qualified Retrieval: Get supplier numbers for suppliers in 'Paris' with STATUS > 20

```
SELECT  S#
FROM    S
WHERE   CITY = 'Paris'
    AND STATUS > 20;
```

RESULT

S#
S_3

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

- Retrieval with Ordering: The result may be ordered based on the contents of one or several fields;

  column [order] [, column [order]] …

- where ‘order’ is either ASC or DESC, and ASC as the default.

Database Systems

- Get supplier numbers and Status for suppliers in 'Paris' in descending order of status.

```
SELECT      S#, STATUS
FROM        S
WHERE       CITY = 'Paris'
ORDER BY STATUS DESC;
```

RESULT

S#	Status
S_3	30
S_2	10

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

Join Queries

- Simple equi-join

```
SELECT   S.*, P.*
FROM     S, P
WHERE    S.CITY = P.CITY;
```

- Note that the field referenced in the WHERE clause here must be qualified by the table names.

- Conceptually, you may generate the Cartesian product of the tables listed in the FROM clause. Then eliminate all the tuples that do not satisfy the join condition defined in WHERE clause.

Database Systems

S

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

P

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

Database Systems

RESULT

S#	Sname	Status	S.City	P#	Pname	Color	Weight	P.City
S_1	Smith	20	London	P_1	Nut	Red	12	London
S_1	Smith	20	London	P_4	Screw	Red	14	London
S_1	Smith	20	London	P_6	Cog	Red	19	London
S_2	Jones	10	Paris	P_2	Bolt	Green	17	Paris
S_2	Jones	10	Paris	P_5	Cam	Blue	12	Paris
S_3	Blake	30	Paris	P_2	Bolt	Green	17	Paris
S_3	Blake	30	Paris	P_5	Cam	Blue	12	Paris
S_4	Clark	20	London	P_1	Nut	Red	12	London
S_4	Clark	20	London	P_4	Screw	Red	14	London
S_4	Clark	20	London	P_6	Cog	Red	19	London

Database Systems

- Greater-than join: Get all combinations of supplier and part information such that the supplier city follows the part city in alphabetical order;

```
SELECT  S.*, P.*
FROM    S, P
WHERE  S.CITY > P.CITY;
```

S

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

P

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

S#	Sname	Status	S.City	P#	Pname	Color	Weight	P.City
S_2	Jones	10	Paris	P_1	Nut	Red	12	London
S_2	Jones	10	Paris	P_4	Screw	Red	14	London
S_2	Jones	10	Paris	P_6	Cog	Red	19	London
S_3	Blake	30	Paris	P_1	Nut	Red	12	London
S_3	Blake	30	Paris	P_4	Screw	Red	14	London
S_3	Blake	30	Paris	P_6	Cog	Red	19	London

Database Systems

- Get all combinations of supplier information and part information where the supplier and part concerned are co-located, but omitting supplier with status 20;

```
SELECT   S.*, P.*
FROM     S, P
WHERE    S.CITY = P.CITY
   AND   STATUS < > 20;
```

S

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

P

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

RESULT

S#	Sname	Status	S.City	P#	Pname	Color	Weight	P.City
S_2	Jones	10	Paris	P_2	Bolt	Green	17	Paris
S_2	Jones	10	Paris	P_5	Cam	Blue	12	Paris
S_3	Blake	30	Paris	P_2	Bolt	Green	17	Paris
S_3	Blake	30	Paris	P_5	Cam	Blue	12	Paris

- Aggregate Functions
  - Aggregate functions are used to enhance the retrieval power of SQL. These are:

COUNT	number of values in the column
SUM	sum of the values in the column
AVG	average of the values in the column
MAX	largest value in the column
MIN	smallest value in the column

- Aggregate Functions
  - For SUM and AVG, column must be numeric values.
  - Key word DISTINCT can be used to eliminate the duplicate values.
  - For COUNT, DISTINCT must be specified.

Database Systems

- Get the total number of suppliers

```
SELECT   COUNT (*)
FROM     S;
```

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

RESULT

5

- Note that the result is a table with a single value.

Database Systems

- Get the total number of suppliers currently supplying part;

```
SELECT   COUNT ( DISTINCT S#)
FROM     SP;
```

RESULT

4

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

Database Systems

- Get the number of shipments for part 'P_2';

```
SELECT   COUNT (*)
FROM     SP
WHERE    P# = 'P2';
```

RESULT

4

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

Database Systems

- Get the total quantity of part ‘P_2’ supplied;

```
SELECT   SUM (QTY)
FROM     SP
WHERE    P# = ‘P2’;
```

RESULT

1,000

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

Database Systems

- So far we have applied aggregate operators to all (qualifying) tuples. Sometimes, it is desirable to apply them to each of several groups of tuples. Assume the following relation:
  - Sailors(sid:integer, sname:string, rating:integer, age:real)
- Further assume we have the following query:
  - Find the age of the youngest sailor for each rating level;

Database Systems

- In general, we do not know how many rating levels exist, and also we do not know what the rating values for these levels are!
- To simplify the situation, suppose we know that rating values go from 1 to 10;

Database Systems

- We can write 10 queries such as:

```
SELECT  MIN (S.age)
FROM    Sailors S
WHERE   S.rating = i;          1 ≤ i ≤ 10
```

- Not a good solution!

Database Systems

- The GROUP BY and HAVING commands can be used to solve the issue.

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

- The target-list consist of:
  - a list of *attribute names*,
  - a list of terms having the form aggregate (*attribute-name*) AS *new-name*.
- Attribute (s) that appeared in *attribute names* must appear in the grouping-list.
- The expression appearing in the group-qualification must have a single value per group.

Database Systems

- Order of Operations:
  - Cartesian product of *relation-list* is performed.
  - Restrictions specified in the *qualification* are applied.
  - Projection is enforced to eliminate unnecessary attributes.
  - The resultant relation is sorted according to *grouping-list*.
  - The *group-qualification* in the HAVING clause is enforced.

- Use of GROUP BY
  - The GROUP BY operator conceptually (logically) rearranges the table represented in FROM clause into partitions, such that within any one group all rows have the same value for the GROUP BY field.

Database Systems

- For each part supplied, get the part number and the total shipment quantity.

```
SELECT      P#, SUM (QTY) AS Total
FROM        SP
GROUP BY    P#;
```

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

RESULT

P#	Total
P_1	600
P_2	1,000
P_3	400
P_4	500
P_5	500
P_6	100

Database Systems

- Use of HAVING
  - Get part numbers for all parts supplied by more than one supplier;

```
SELECT      P#
FROM        SP
GROUP BY    P#
HAVING      COUNT (*) > 1;
```

RESULT

P#
P_1
P_2
P_4
P_5

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

Database Systems

- Nested Queries
  - A nested query is a query that has another query embedded within it; the embedded query is called a sub-query. A sub-query typically appears in the WHERE clause. The sub-query may appear in FROM clause or HAVING clause, as well.

Database Systems

- Get supplier names of suppliers who supply part 'P_2';

```
SELECT   Sname
FROM     S
WHERE    S#  IN
         ( SELECT  S#
           FROM    SP
           WHERE   P# = 'P2' ) ;
```

- The overall query is evaluated by evaluating the nested part first.

Database Systems

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

RESULT

Sname
Smith
Jones
Blake
Clark

- This query is equivalent to:

```
SELECT   Sname
FROM     S
WHERE    S#  IN ('S1', 'S2', 'S3', 'S4') ;
```

- This can also be expressed as a join query

```
SELECT   Sname
FROM     S, SP
WHERE    S.S# = SP.S#
    AND  SP.P# = 'P2';
```

Database Systems

- Multiple levels of nesting
  - Got supplier names for suppliers who supply at least one 'red' part;

```
SELECT   Sname
FROM     S
WHERE    S#  IN
         ( SELECT   S#
           FROM     SP
           WHERE    P#  IN
                    ( SELECT   S#
                      FROM     P
                      WHERE    COLOR = 'Red'));
```

Database Systems

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

RESULT

Sname
Smith
Jones
Clark

Database Systems

- Aggregate function in a sub-query
  - Get supplier numbers for supplies with status value less than the current maximum status value in the *S* table;

```
SELECT   S#
FROM     S
WHERE    STATUS <
         ( SELECT   MAX (STATUS)
           FROM     S);
```

Database Systems

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

RESULT

S#
S_1
S_2
S_4

Query Using EXISTS

- EXISTS is one of the most fundamental and general constructs in SQL language.

Database Systems

- Get supplier names for suppliers who supply part ‘P_2’;

```
SELECT   Sname
FROM     S
WHERE    EXISTS
         ( SELECT    *
           FROM      SP
           WHERE     S# = S.S#
               AND   P# = ‘P2’ ) ) ;
```

Database Systems

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

RESULT

SNAME
Smith
Jones
Blake
Clark

To see how the example works, consider each Sname in turn and see whether it causes the existence test to evaluate to True.

Database Systems

- Get supplier names for suppliers who do not supply part 'P_2' (inverse of the previous example);

```
SELECT   Sname
FROM     S
WHERE    NOT EXISTS
     ( SELECT    *
       FROM      SP
       WHERE     S# = S.S#
          AND    P# = 'P2' ) ) ;
```

Database Systems

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

RESULT

Sname
Adams

Database Systems

- Last query can also be represented by using negated form of IN;

```
SELECT   Sname
FROM     S
WHERE    S#   NOT IN
        ( SELECT   S#
          FROM     SP
          WHERE    P# = 'P2' ) ;
```

Database Systems

- Get supplier names for suppliers who supply all parts;

```
SELECT   Sname
FROM     S
WHERE    NOT EXISTS
      ( SELECT   *
        FROM     P
        WHERE    NOT EXISTS
              ( SELECT   *
                FROM     SP
                WHERE    S# = S.S#
                   AND   P# = P.P# ) ) ;
```

- The previous query can be expressed as:
  - Select supplier names for suppliers such that there does not exist a part that they do not supply.

Database Systems

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

RESULT

Sname
Smith

Database Systems

- **Query Using Union:** Union is traditional union operator borrowed from set theory.
  - Get supplier numbers for parts that either weight more than 16 Pounds or are supplied by supplier ‘S_2’.

Database Systems

```
SELECT   P#
FROM     P
WHERE    WEIGHT > 16
UNION
SELECT   P#
FROM     SP
WHERE    S# = 'S2' ;
```

- Note redundant duplicate rows are always eliminated. However, we can use UNION ALL operator to include the duplicates.

Database Systems

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

RESULT

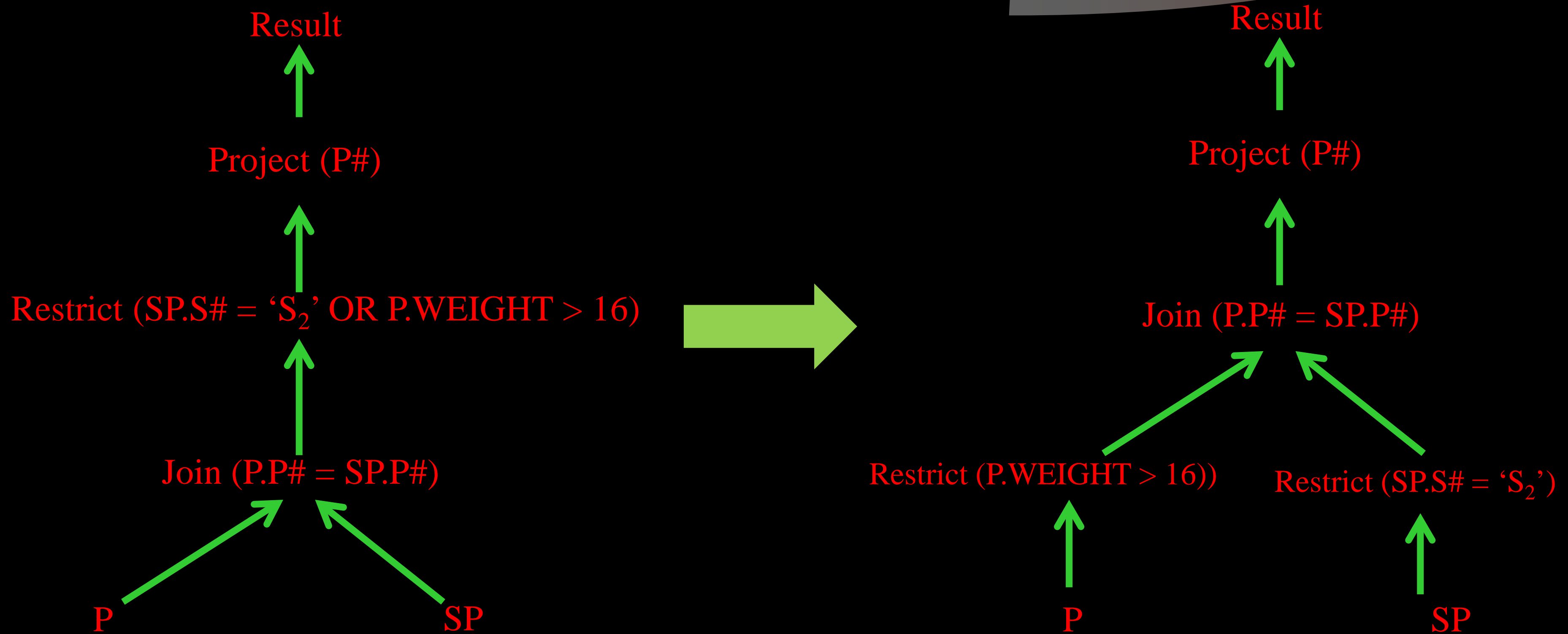
P#
P_1
P_2
P_3
P_6

Database Systems

- Previous query can also be written as:

```
SELECT    DISTINCT P#
FROM      P, SP
WHERE     P.P# = SP.P#
AND       P.WEIGHT > 16
OR        SP.S# = 'S2';
```

Database Systems



Database Systems

- Query Using INTERSECT: Similarly INTERSECT operator has also been borrowed from traditional set theory:

```
SELECT   P#
FROM     P
WHERE    WEIGHT > 16
INTERSECT
SELECT   P#
FROM     SP
WHERE    S# = 'S2';
```

Database Systems

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

RESULT

P#
P_2

Database Systems

- Previous query can also be specified as:

```
SELECT   P#
FROM     P, SP
WHERE    P.P# = SP.P#
AND      P.WEIGHT > 16
AND      SP.S# = 'S2') ;
```

- Modification Operations
  - The SQL DML supports three modification operations:
    - UPDATE
    - DELETE
    - INSERT
  - Note these operations change the contents of the database, hence they may violate the integrity constraints.

Database Systems

- UPDATE: The general format of UPDATE operation is;

```
UPDATE      table
SET         field = scalar-expression
            [, field = scalar-expression] …
[WHERE       condition] ;
```

- All records in *table* satisfying *condition* are modified in accordance with the *assignments*.

Database Systems

- Change the color of part ‘P_2’ to yellow, increase its weight by 5, and set its city to unknown (null);

```
UPDATE    P
SET       COLOR = 'Yellow'
          WEIGHT = WEIGHT + 5,
          CITY = NULL
WHERE     P# = 'P2';
```

Database Systems

- Double the status of all suppliers in 'London';

```
UPDATE     S
SET        STATUS = STATUS * 2
WHERE      CITY = 'London' ;
```

Database Systems

- Set the shipment quantity to zero for all supplies in ‘London’;

```
UPDATE      SP
SET         QTY = 0
WHERE       'London' =
            ( SELECT    CITY
              FROM      S
              WHERE     S.S# = SP.S# ) ;
```

Database Systems

- Update Multiple Tables:

```
UPDATE    S
SET       S# = 'S9'
WHERE     S# = 'S2'

UPDATE    SP
SET       S# = 'S9'
WHERE     S# = 'S2'
```

- The first UPDATE will force the database to become inconsistent, since now in *shipment* table there is a *supplier* 'S_2' that does not exist. The database remains in inconsistent state until after the second UPDATE is executed.

Database Systems

- DELETE: This command has the following general format;

```
DELETE
FROM      table
[WHERE    condition];
```

- All records in *table* satisfying *condition* are deleted.

Database Systems

- Delete all shipments with quantity greater than 300;

```
DELETE
FROM     SP
WHERE    QTY > 300;
```

- Delete supplier ‘S_5’;

```
DELETE
FROM     S
WHERE    S# = ‘S5’;
```

- Delete all shipments;

```
DELETE
FROM      SP ;
```

- Note that now *SP* is an empty table.
- Delete all shipments for suppliers in 'London';

```
DELETE
FROM      SP
WHERE     'London' =
    ( SELECT  CITY
      FROM    S
      WHERE   S.S# = SP.S# ) ;
```

Database Systems

- INSERT: Insert comes in two formats;

```
INSERT
INTO        table [ (field [, field ] … ) ]
VALUE       (literal [, literal ] … ) ;
```

- In this case a record with the contents defined in VALUE clause is added to the *table*.

```
INSERT
INTO        table [ (field [, field ] … ) ]
sub-query ;
```

- In this case, the result of the sub-query (may be multiple rows) is added to the *table*.
- In both cases, omitting the list of fields means all fields in the table, in left to right order.

Database Systems

- Add part ‘P_7’ (city ‘Athens’, weight 24) name and color unknown to the *P* relation;

```
INSERT
INTO        P ( P#, CITY, WEIGHT)
VALUE       ( 'P7', 'Athens', 24) ;
```

- Note we assumed that *COLOR* and *Pname* are not defined as ‘NOT NULL’.

Database Systems

- Add part 'P_8' (name 'Sprocket', color 'Pink', city 'Nice', weight 14) to the *P* relation;

```
INSERT
INTO     P
VALUE    ( 'P8', 'Sprocket', 'Pink', 14, 'Nice' ) ;
```

- Add a new shipment with supplier number 'S_{20}', part number 'P_{20}' and quantity 1000.

```
INSERT
INTO     SP (S#, P#, QTY)
VALUE    ( 'S20', 'P20', 1000 ) ;
```

- Join Types and Conditions
  - Each of the variant of the join operations in SQL consists of a join type and a join condition.
  - Join condition defines which tuples in the two relations match and what attributes are present in the join result.
  - Join type defines how tuples in each relation that do not match any tuple in the other relation are treated.

Database Systems

Join Types and Conditions

Join Type
Inner join
Left outer join
Right outer join
Full outer join

Join Conditions
Natural
On <predicate>
Using (A_1, A_2, …, A_n)

- Use of a join condition is mandatory for outer join and optional for inner join.
- Keyword natural appears before join type, whereas on and using conditions appear at the end of join expression.

- Join Types and Conditions
  - The ordering of the attributes in the result of a natural join is as follows:
    - Join attributes appears first in the same order as they are in the left hand side relation,
    - Nonjoin attributes of left hand side relation,
    - Nonjoin attributes of right hand side relation.

Database Systems

- Join Types and Conditions
  - The right outer join is symmetric to the left outer join.
  - Tuples from the right hand side relation that do not match any tuples in the left hand side relation are padded with nulls and added to the result relation.

Database Systems

- Join Types and Conditions
  - The join condition using (A_1, A_2, …, A_n) is similar to natural join condition, except that the join attributes are A_1, A_2, …, A_n, rather than all common attributes. In addition, join attributes A_1, A_2, …, A_n appear just once in the join result.

Database Systems

- Example
  - Assume the following relations

loan

Loan-number	Branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

Customer-name	Loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

Database Systems

- Example
  - loan inner join borrower on loan.loan-number = borrower.loan-number

Loan-number	Branch-name	Amount	Customer-name	Loan-number
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

  - loan left outer join borrower on loan.loan-number = borrower.loan-number

Loan-number	Branch-name	Amount	Customer-name	Loan-number
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	Null	null

Database Systems

- Example
  - loan natural right outer join borrower

Loan-number	Branch-name	Amount	Customer-name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	null	null	Hayes

  - loan full outer join borrower using (loan-number)

Loan-number	Branch-name	Amount	Customer-name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	null
L-155	null	null	Hayes

Database Systems

- What is the result of the following?

```
INSERT
INTO        account
            SELECT    *
            From      account
```

Database Systems

- For each part supplied, get the part number and the total quantity supplied supported for that part and save the result in the database;

```
CREATE   TABLE          TEMP
         ( P#           CHAR (6)    NOT NULL,
           TOTQTY       INTEGER     NOT NULL,
         PRIMARY KEY    (P# )) ;
```

Database Systems

```
INSERT
INTO      TEMP    (P#, TOTQTY )
          SELECT     (P#, SUM (QTY)
          FROM        P#
          GROUP BY   P# ;
```

- SELECT is executed and result is copied in *Temp* relation. User, now, can do whatever he/she wants to do with *Temp* relation. Eventually,

```
DROP   TABLE   TEMP ;
```

will eliminate *Temp* relation from the database.

Database Systems

- Questions: Using the following relations;

S

S#	Sname	Status	City
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

P

P#	Pname	Color	Weight	City
P_1	Nut	Red	12	London
P_2	Bolt	Green	17	Paris
P_3	Screw	Blue	17	Rome
P_4	Screw	Red	14	London
P_5	Cam	Blue	12	Paris
P_6	Cog	Red	19	London

SP

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

Database Systems

- Get supplier names for suppliers who supply part P_2.
- Get supplier names for suppliers who supply at least one red part.
- Get supplier names for suppliers who supplies all parts.
- Get supplier numbers for suppliers who supply at least all those parts supplied by supplier S_2.
- Get supplier names for suppliers who do not supply part P_2.
- Get all pairs of supplier numbers such that the two suppliers concerned are co-located