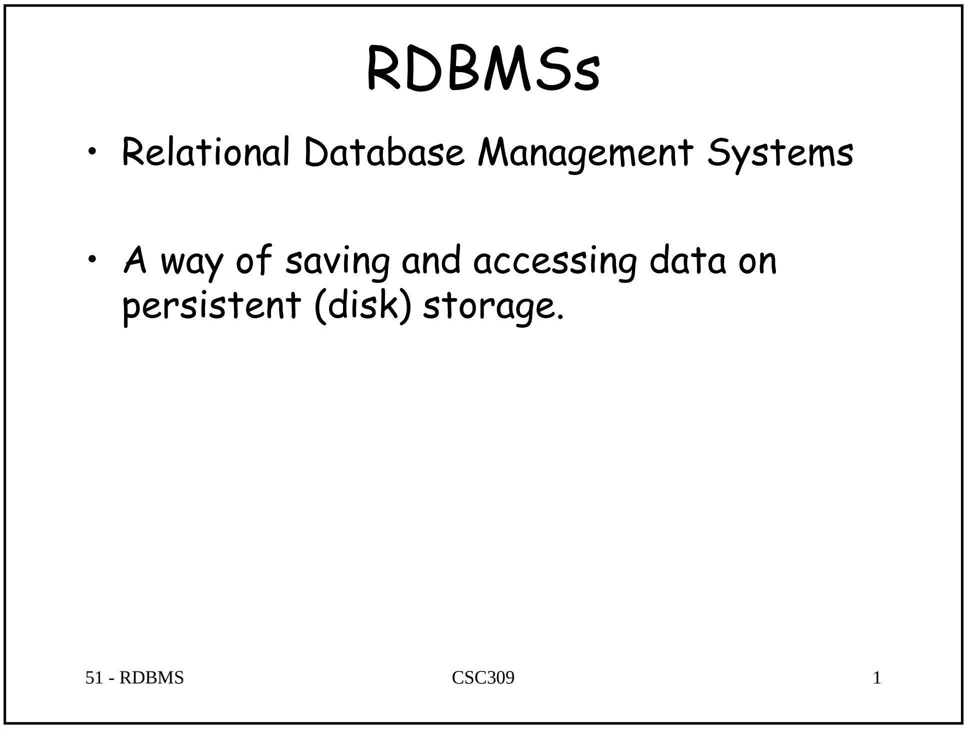
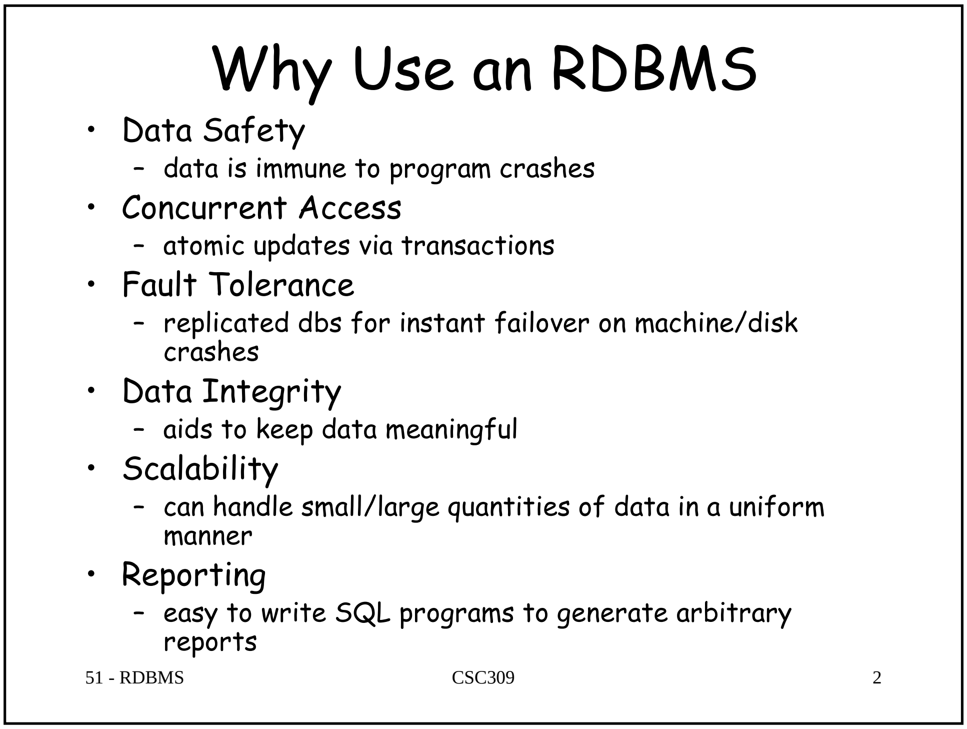


RDBMSs

- Relational Database Management Systems
- A way of saving and accessing data on persistent (disk) storage.



Why Use an RDBMS

- Data Safety
  - data is immune to program crashes
- Concurrent Access
  - atomic updates via transactions
- Fault Tolerance
  - replicated dbs for instant failover on machine/disk crashes
- Data Integrity
  - aids to keep data meaningful
- Scalability
  - can handle small/large quantities of data in a uniform manner
- Reporting
  - easy to write SQL programs to generate arbitrary reports

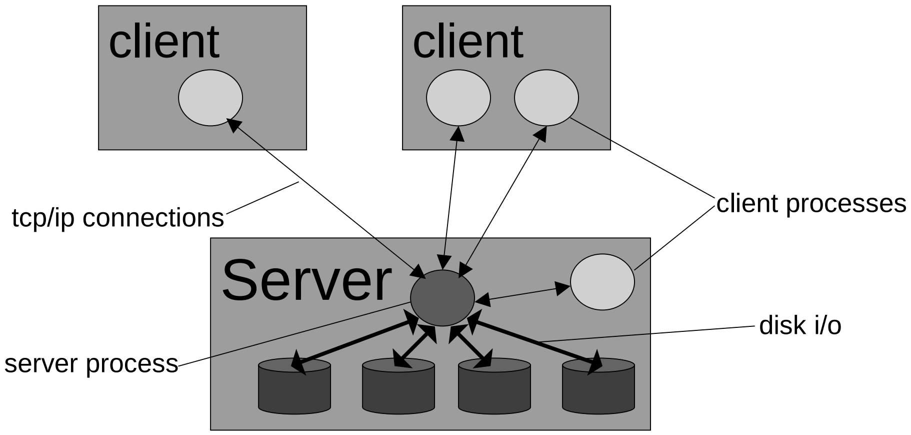
Relational Model

- First published by E.F. Codd in 1970
- A relational database consists of a collection of tables
- A table consists of rows and columns
- each row represents a record
- each column represents an attribute of the records contained in the table

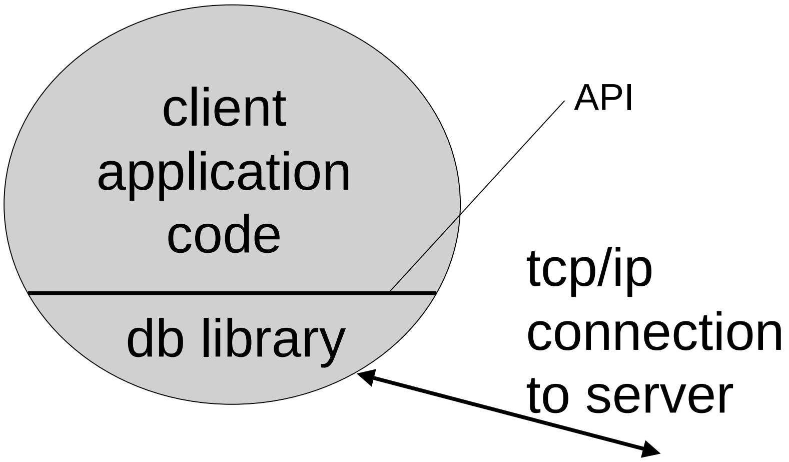
RDBMS Technology

- Client/Server Databases
  - Oracle, Sybase, MySQL, SQLServer
- Personal Databases
  - Access
- Embedded Databases
  - Pointbase

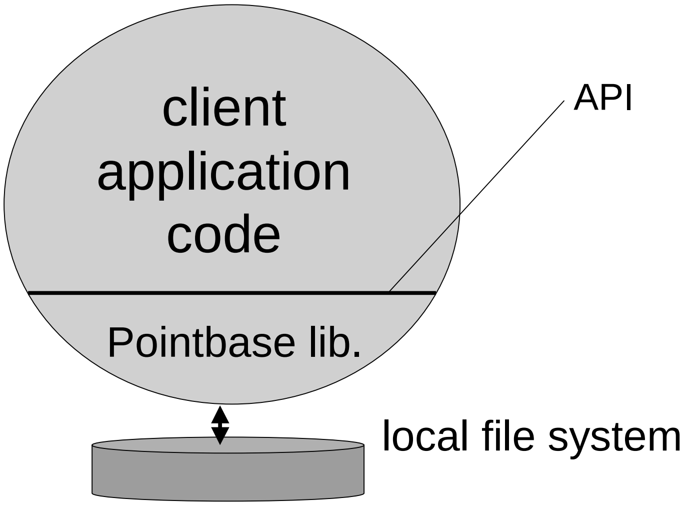
Client/Server Databases



Inside the Client Process



Pointbase



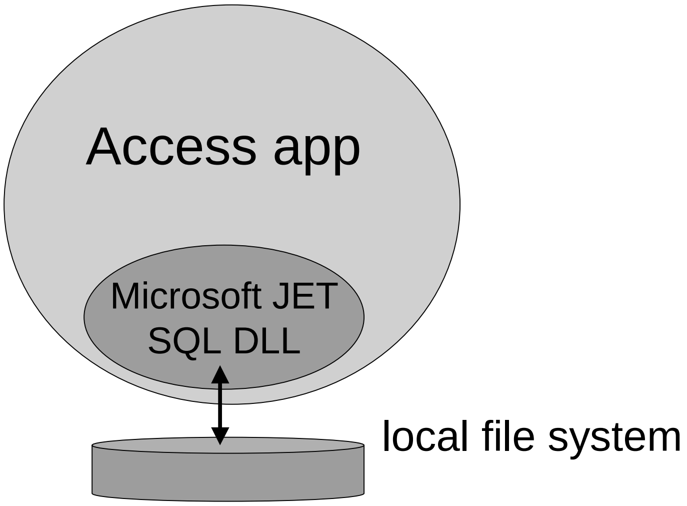
client application code

API

Pointbase lib.

local file system

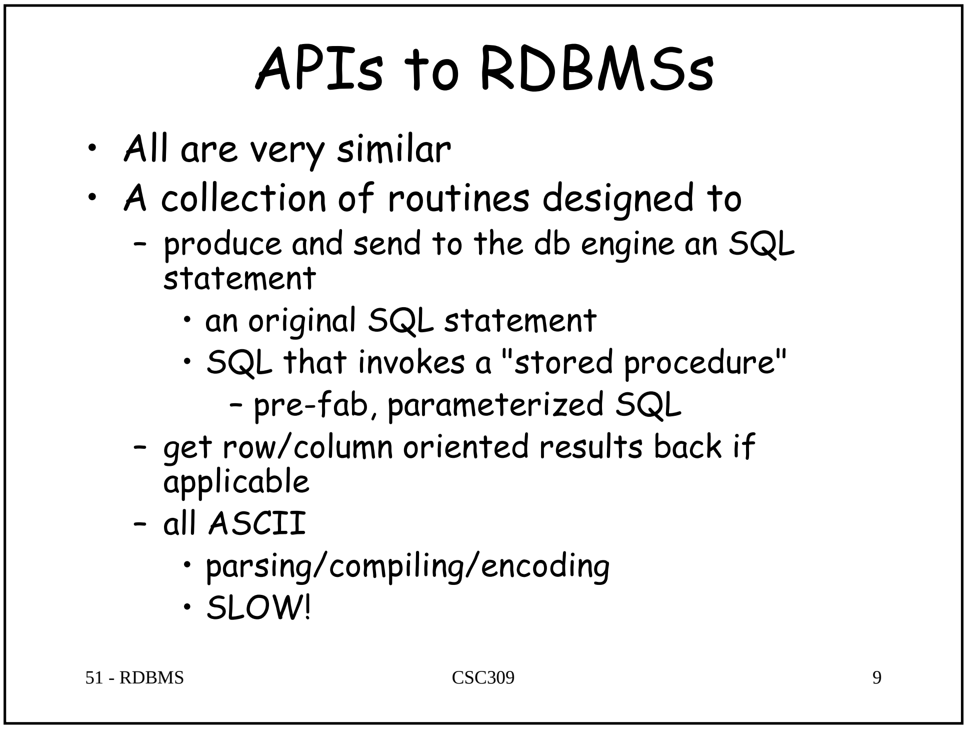
Microsoft Access



Access app

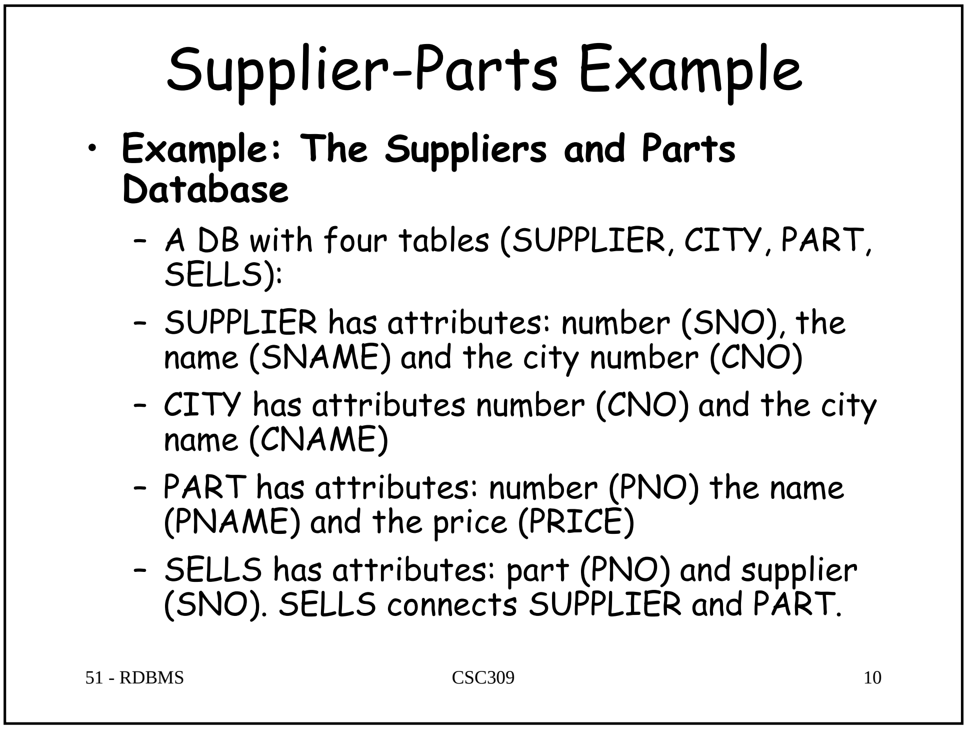
Microsoft JET SQL DLL

local file system



APIs to RDBMSs

- All are very similar
- A collection of routines designed to
  - produce and send to the db engine an SQL statement
    - an original SQL statement
    - SQL that invokes a "stored procedure"
      - pre-fab, parameterized SQL
  - get row/column oriented results back if applicable
  - all ASCII
    - parsing/compiling/encoding
    - SLOW!



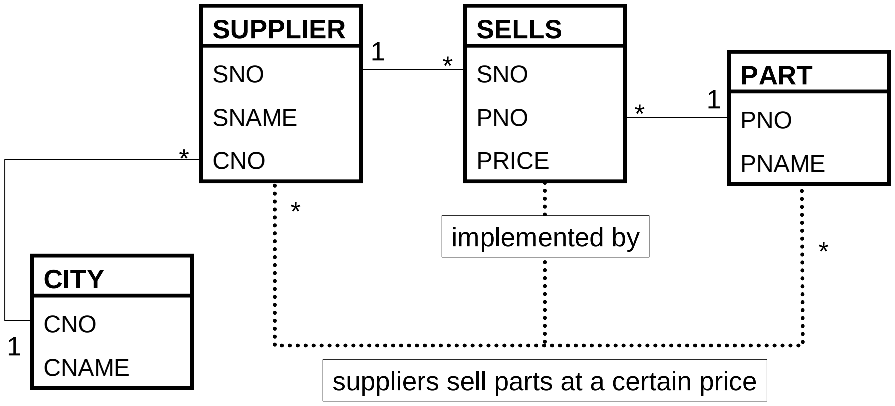
Supplier-Parts Example

- **Example: The Suppliers and Parts Database**
  - A DB with four tables (SUPPLIER, CITY, PART, SELLS):
  - SUPPLIER has attributes: number (SNO), the name (SNAME) and the city number (CNO)
  - CITY has attributes number (CNO) and the city name (CNAME)
  - PART has attributes: number (PNO) the name (PNAME) and the price (PRICE)
  - SELLS has attributes: part (PNO) and supplier (SNO). SELLS connects SUPPLIER and PART.

Entities and Relations

- An *entity* represents something real
- A *relation* represents a connection between entities
- The tables PART and SUPPLIER may be regarded as *entities*
- SELLS may be regarded as a *relationship* between a particular part and a particular supplier.

Supplier-Parts E/R



SUPPLIER
SNO
SNAME
CNO

SELLS
SNO
PNO
PRICE

PART
PNO
PNAME

CITY
CNO
CNAME

1
*
*
1
*
1
*
*

implemented by

suppliers sell parts at a certain price

Supplier-Parts Example

```
CITY:
CNO |  CNAME
----+--------
 1  | London
 2  | Paris
 3  | Rome
 4  | Vienna
```

Row `(1, London)` in CITY represents a distinct city, London, with city number 1.

eg.mdb

Supplier-Parts Example

```
SUPPLIER:
SNO |  SNAME   | CNO
----+---------+-----
 1  |  Smith  |  1
 2  |  Jones  |  2
 3  |  Adams  |  1
 4  |  Blake  |  3
```

Row `(1,Smith, 1)` in SUPPLIER represents a supplier with supplier number 1 whose name is Smith and who is based in the city numbered 1 (London).

Supplier-Parts Example

```
PART:
PNO |  PNAME
----+--------
 1  |  Screw
 2  |  Nut
 3  |  Bolt
 4  |  Cam
```

Row `(2,Nut)` in PART represents a part with part number 2, and part name Nut.

Supplier-Parts Example

```
SELLS:
 SNO | PNO | PRICE
-----+-----+-------
  1  |  1  |   10
  1  |  2  |    8
  2  |  4  |   38
  3  |  1  |   11
  3  |  3  |    6
  4  |  2  |    7
  4  |  3  |    4
  4  |  4  |   45
```

Row `(1,2,8)` in SELLS represents the relationship of supplier 1 (Smith) selling part 2 (Nut)

for 10 cents.

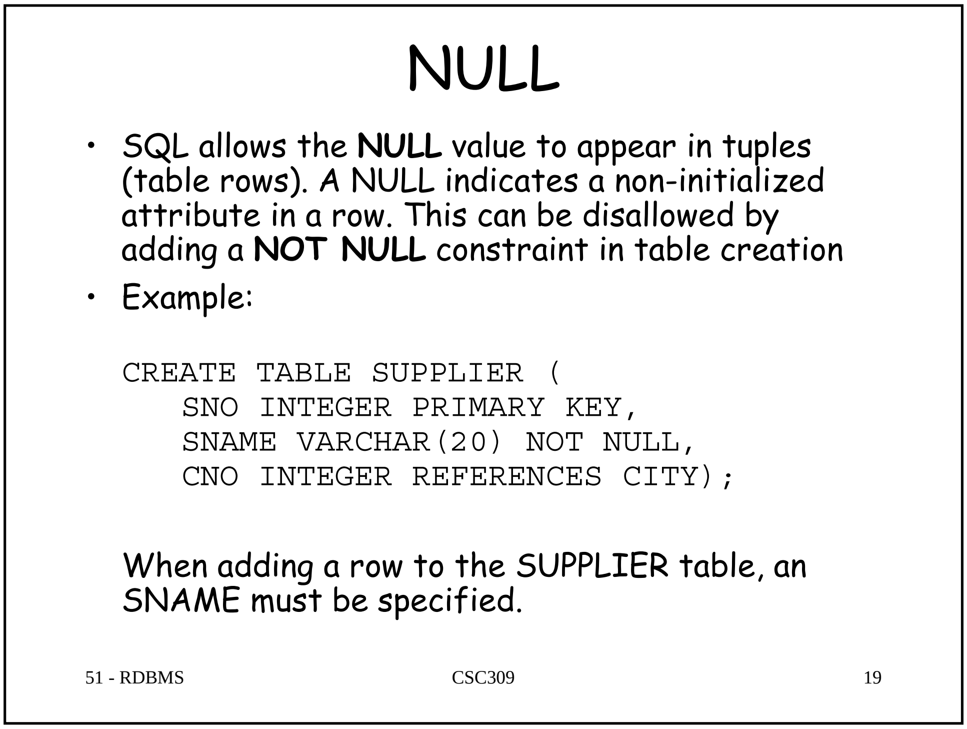
Structured Query Language

- **Structured Query Language (SQL)**
  - Used on a RDBMS to create, search and modify tables.
  - Table creation (by example): Executing the following causes the creation of the Suppliers and Parts database above.
- Example:

```
CREATE TABLE CITY
    (CNO INTEGER PRIMARY KEY, CNAME VARCHAR(20));
CREATE TABLE SUPPLIER
    (SNO INTEGER PRIMARY KEY,
     SNAME VARCHAR(20), CNO INTEGER REFERENCES CITY);
CREATE TABLE PART
    (PNO INTEGER PRIMARY KEY, PNAME VARCHAR(20));
CREATE TABLE SELLS
    (SNO INTEGER REFERENCES SUPPLIER,
     PNO INTEGER REFERENCES PART,
     PRICE DECIMAL(4,2),
     PRIMARY KEY (SNO, PNO));
```

SQL Datatypes

- Some SQL Data types your database may support
  - INTEGER: signed fullword binary integer
  - DECIMAL (p[,q]): signed packed decimal number of up to p digits, with q digits to the right of the decimal point. If q is omitted it is assumed to be 0. MSAccess supports NUMBER instead of DECIMAL
  - FLOAT: signed doubleword floating point number.
  - CHAR(n): fixed length character string of length n.
  - VARCHAR(n):varying length character string of maximum length n
  - DATE: A date attribute in a DBMS-specific format.

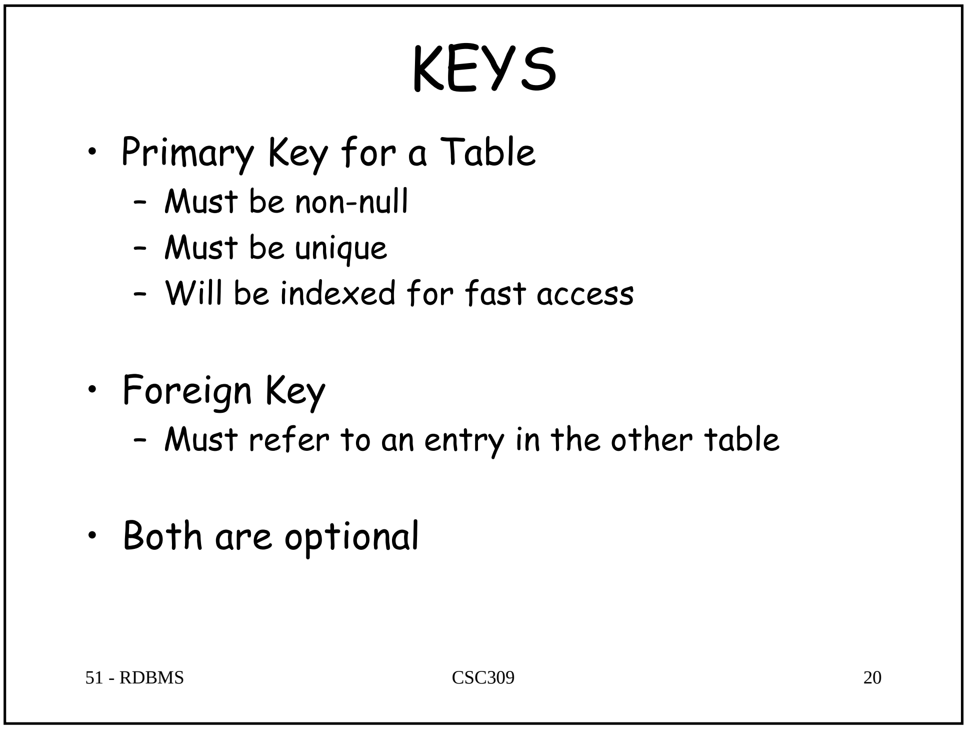


NULL

- SQL allows the **NULL** value to appear in tuples (table rows). A NULL indicates a non-initialized attribute in a row. This can be disallowed by adding a **NOT NULL** constraint in table creation
- Example:

```
CREATE TABLE SUPPLIER (
   SNO INTEGER PRIMARY KEY,
   SNAME VARCHAR(20) NOT NULL,
   CNO INTEGER REFERENCES CITY);
```

When adding a row to the SUPPLIER table, an SNAME must be specified.



KEYS

- Primary Key for a Table
  - Must be non-null
  - Must be unique
  - Will be indexed for fast access
- Foreign Key
  - Must refer to an entry in the other table
- Both are optional

Searching Using SQL

The result of a SQL search is a table.

(Query1) `SELECT * FROM PART;`

(Query2)

```
SELECT * FROM SELLS WHERE PRICE > 11;
```

Searching Using SQL

(Query3)

```
SELECT SNO,PRICE FROM SELLS WHERE
PRICE > 11;
```

(Query4)

```
SELECT PNO, PRICE FROM SELLS WHERE
PNO = 1 AND PRICE <= 10;
```

Searching Using SQL

Cartesian products

(Query5)

```
SELECT * FROM SUPPLIER, PART;
```

The above example forms the table SUPPLIER x PART with rows of the form (s,p) where s is a row in SUPPLIER, p is a row in PART.

In total SUPPLIER x PART has a total of 4*4 rows.

Searching Using SQL

Joins

Matching up rows in tables based on the same value for columns.

For each supplier, we want the name of their city

(Query 7.1)

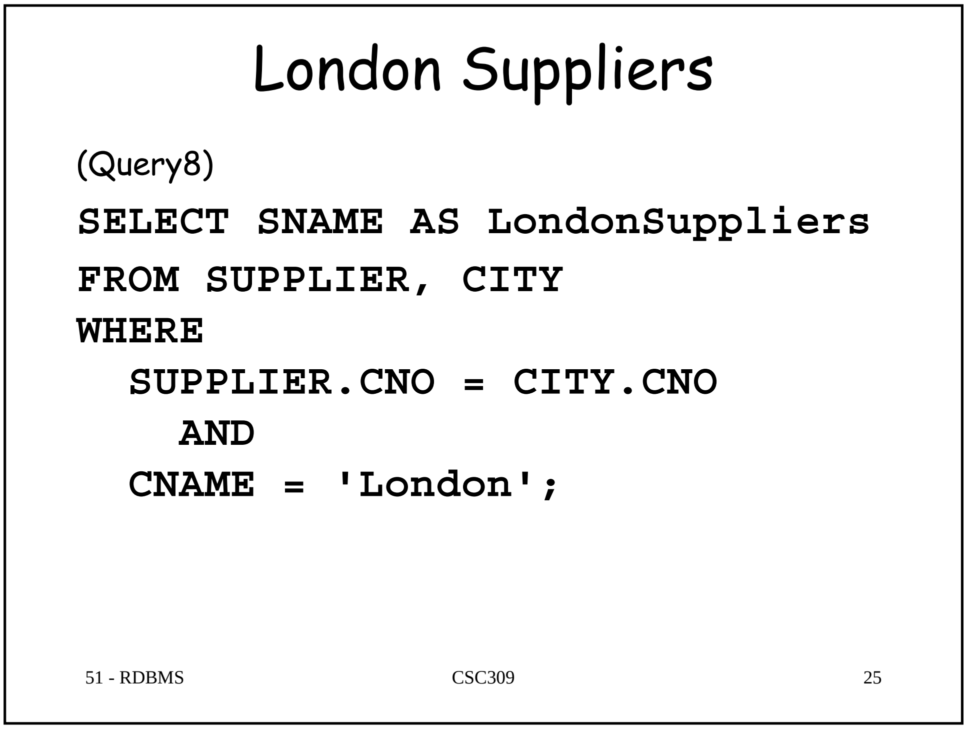
```
SELECT SNAME, CNO FROM SUPPLIER
```

(Query7.2)

```
SELECT *
FROM SUPPLIER AS S, CITY AS C;
```

(Query7.3)

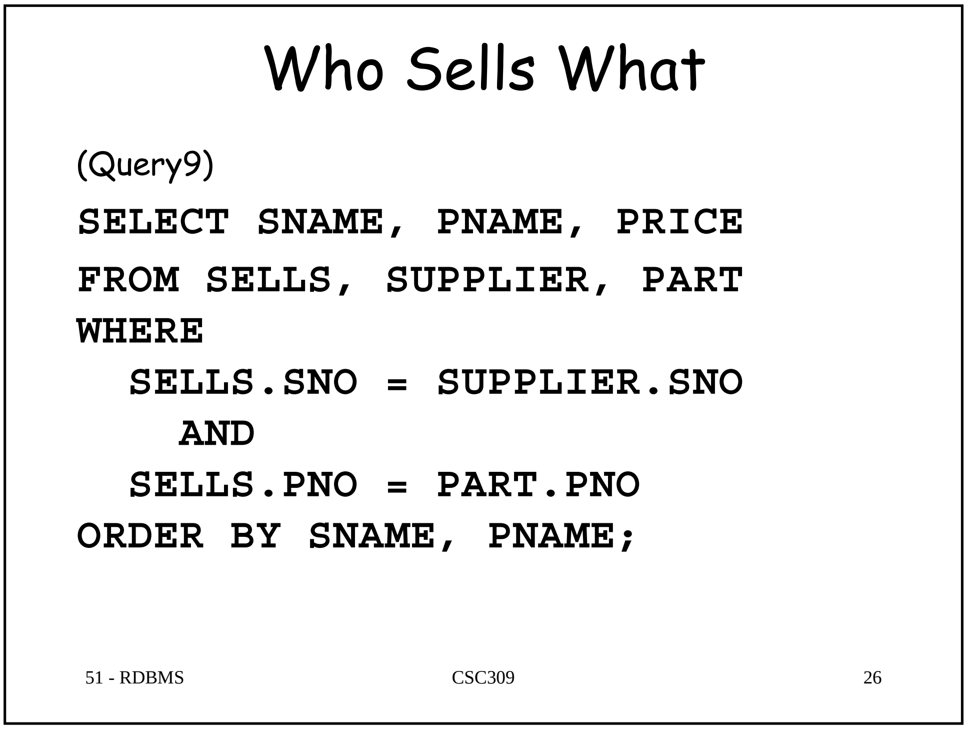
```
SELECT S.SNAME, C.CNAME
FROM SUPPLIER AS S, CITY AS C
WHERE S.CNO = C.CNO;
```



London Suppliers

(Query8)

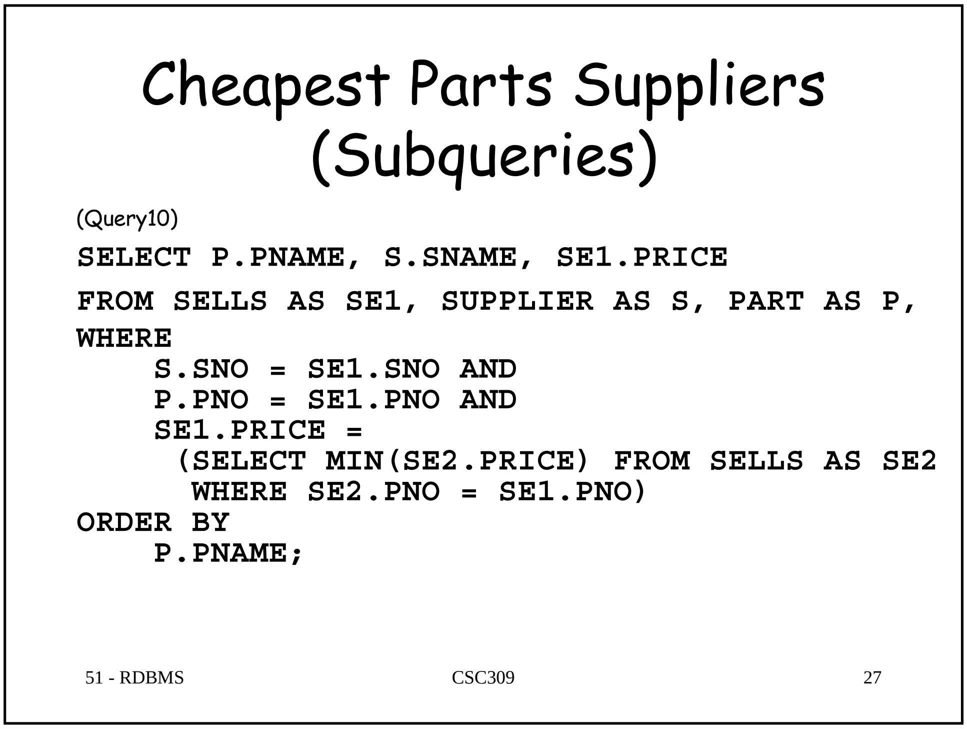
```
SELECT SNAME AS LondonSuppliers
FROM SUPPLIER, CITY
WHERE
  SUPPLIER.CNO = CITY.CNO
    AND
  CNAME = 'London';
```



Who Sells What

(Query9)

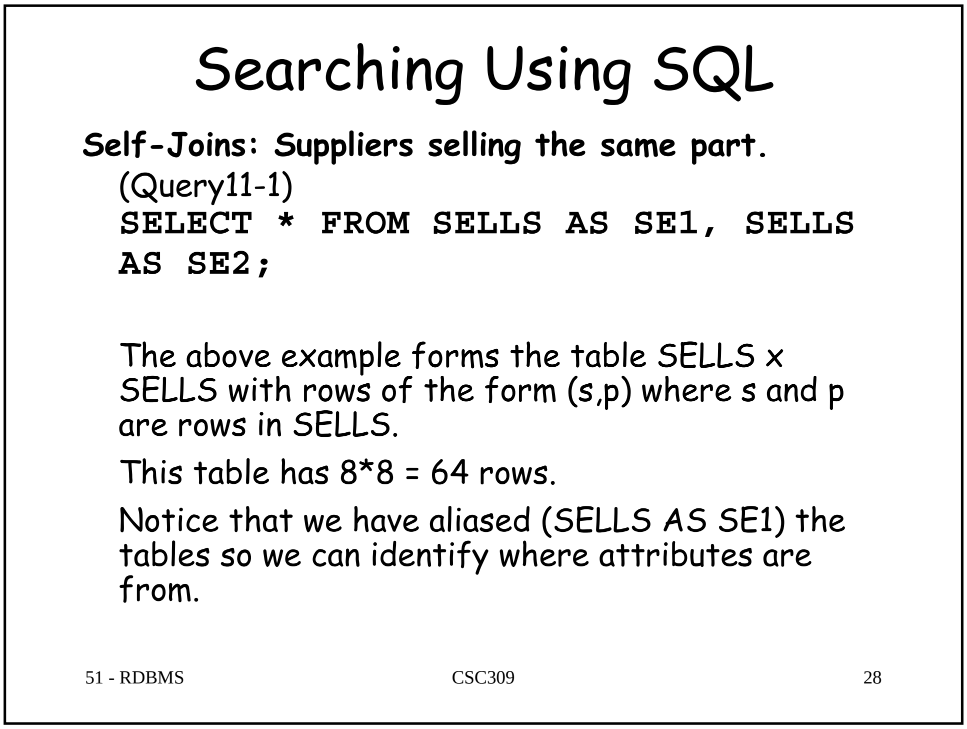
```
SELECT SNAME, PNAME, PRICE
FROM SELLS, SUPPLIER, PART
WHERE
  SELLS.SNO = SUPPLIER.SNO
    AND
  SELLS.PNO = PART.PNO
ORDER BY SNAME, PNAME;
```



Cheapest Parts Suppliers (Subqueries)

(Query10)

```
SELECT P.PNAME, S.SNAME, SE1.PRICE
FROM SELLS AS SE1, SUPPLIER AS S, PART AS P,
WHERE
    S.SNO = SE1.SNO AND
    P.PNO = SE1.PNO AND
    SE1.PRICE =
     (SELECT MIN(SE2.PRICE) FROM SELLS AS SE2
      WHERE SE2.PNO = SE1.PNO)
ORDER BY
    P.PNAME;
```



Searching Using SQL

Self-Joins: Suppliers selling the same part.

(Query11-1)

```
SELECT * FROM SELLS AS SE1, SELLS
AS SE2;
```

The above example forms the table SELLS x SELLS with rows of the form (s,p) where s and p are rows in SELLS.

This table has 8*8 = 64 rows.

Notice that we have aliased (SELLS AS SE1) the tables so we can identify where attributes are from.

Searching Using SQL

Suppliers selling the same part.

(Query11-2)

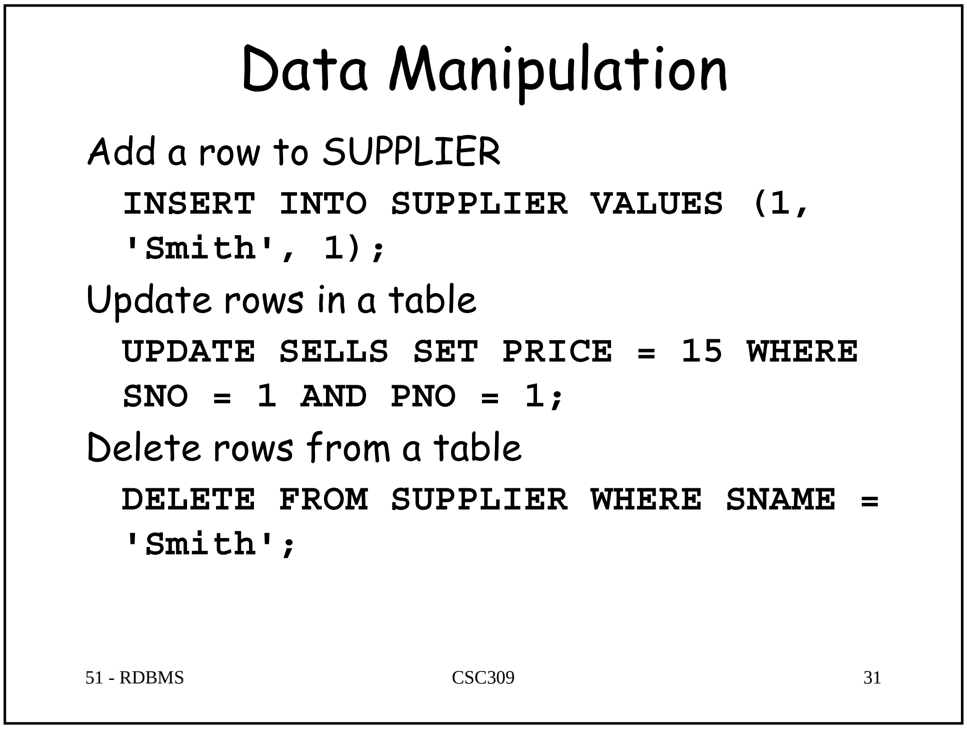
```
SELECT
  SE1.PNO, SE1.SNO, SE2.SNO
FROM
  SELLS AS SE1,
  SELLS AS SE2
WHERE
  SE1.PNO = SE2.PNO AND
  SE1.SNO < SE2.SNO;
```

Searching Using SQL

Pretty-printed:

(Query11-3)

```
SELECT
  PNAME, S1.SNAME, S2.SNAME
FROM
  SELLS AS SE1, SELLS AS SE2, PART,
  SUPPLIER AS S1, SUPPLIER AS S2
WHERE
  SE1.PNO = SE2.PNO AND SE1.SNO < SE2.SNO
   AND SE1.PNO = PART.PNO AND
   SE1.SNO = S1.SNO AND SE2.SNO = S2.SNO
ORDER BY PNAME;
```



Data Manipulation

Add a row to SUPPLIER

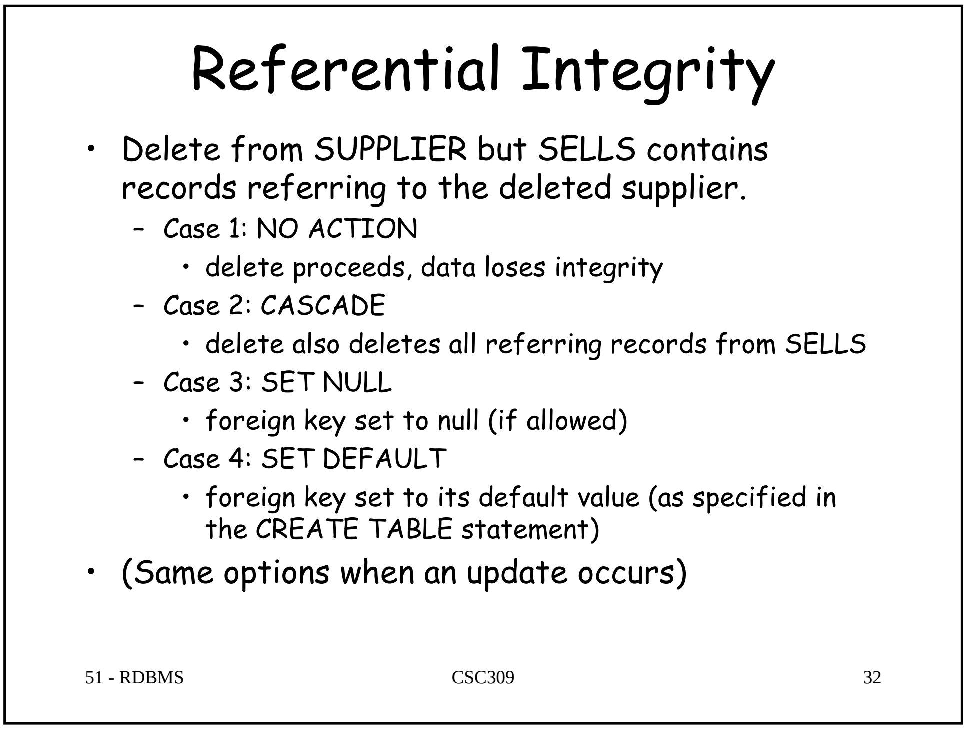
```
INSERT INTO SUPPLIER VALUES (1,
'Smith', 1);
```

Update rows in a table

```
UPDATE SELLS SET PRICE = 15 WHERE
SNO = 1 AND PNO = 1;
```

Delete rows from a table

```
DELETE FROM SUPPLIER WHERE SNAME =
'Smith';
```



Referential Integrity

- Delete from SUPPLIER but SELLS contains records referring to the deleted supplier.
  - Case 1: NO ACTION
    - delete proceeds, data loses integrity
  - Case 2: CASCADE
    - delete also deletes all referring records from SELLS
  - Case 3: SET NULL
    - foreign key set to null (if allowed)
  - Case 4: SET DEFAULT
    - foreign key set to its default value (as specified in the CREATE TABLE statement)
- (Same options when an update occurs)

Referential Integrity

```
CREATE TABLE SELLS(
    SNO
        INTEGER
        NOT NULL
        REFERENCES SUPPLIER
             ON UPDATE CASCADE
             ON DELETE CASCADE,
    PRICE
        NUMBER,
    PNO
        INTEGER
        NOT NULL
        REFERENCES PART
            ON UPDATE CASCADE
            ON DELETE CASCADE);
```

Creating Indices

- Index: a data structure that assists in
  - quickly finding specific rows based on the value of a field,
  - traversing rows in a table on filed-sorted sorted order
- May DB products will automatically create indices on PRIMARY KEY and UNIQUE columns.
- Can ensure and/or create indices on other columns (or column combinations) as well

```
    CREATE INDEX priceIndex ON
           SELLS(PRICE);
```

- Never need an index. For performance reasons only.
- Query optimizer will attempt to make use of indices to optimize the implementation of a given query.

Normal Form

- A database schema with no duplicated information is in *normal form*
  - Good to maintain data integrity
  - Bad for queries/updates
    - slows them down
    - makes them more difficult to write

De-Normalization

```
SUPPLIER (normalized):
SNO |  SNAME   | CNO
----+---------+-----
 1  |  Smith   |  1
 2  |  Jones   |  2
 3  |  Adams   |  1
 4  |  Blake   |  3
```

```
SUPPLIER (de-normalized):
SNO |  SNAME   |  CITY
----+---------+---------
 1  |  Smith   |  London
 2  |  Jones   |  Paris
 3  |  Adams   |  Lodnon
 4  |  Blake   |  Rome
```

De-Normalized Query

```
SUPPLIER (de-normalized):
SNO |   SNAME   |   CITY
----+---------+---------
 1  |   Smith   |   London
 2  |   Jones   |   Paris
 3  |   Adams   |   Lodnon
 4  |   Blake   |   Rome
```

- SELECT SNAME FROM SUPPLIER WHERE CITY = 'London';

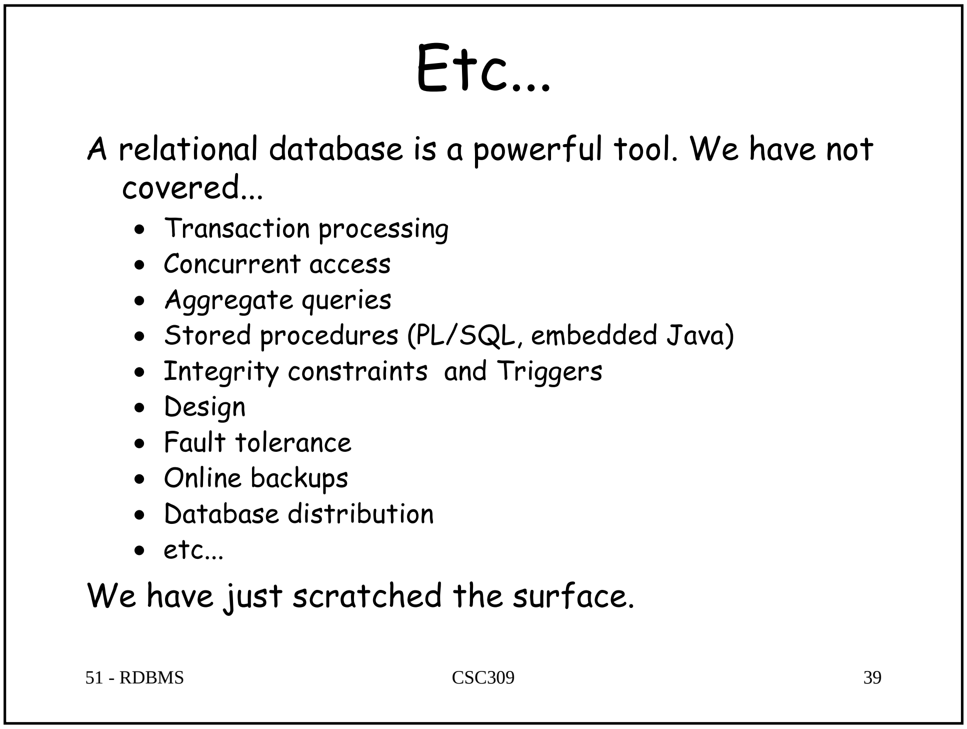
```
SNAME
-----
Smith
```

Normalized Query

```
SUPPLIER (normalized):
SNO |   SNAME   | CNO
----+---------+-----
 1  |   Smith   |  1
 2  |   Jones   |  2
 3  |   Adams   |  1
 4  |   Blake   |  3
```

```
SELECT SNAME
FROM SUPPLIER, CITY
WHERE
  SUPPLIER.CNO = CITY.CNO
   AND
  CITY = 'London';
```

```
SNAME
-----
Smith
Adams
```



Etc...

A relational database is a powerful tool. We have not covered...

- Transaction processing
- Concurrent access
- Aggregate queries
- Stored procedures (PL/SQL, embedded Java)
- Integrity constraints and Triggers
- Design
- Fault tolerance
- Online backups
- Database distribution
- etc...

We have just scratched the surface.