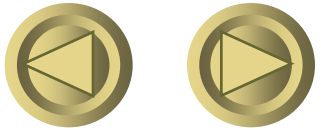
Chapter 6 The database Language SQL –as a tutorial

- **About SQL**

  SQL is a standard database language, adopted by many commercial systems.

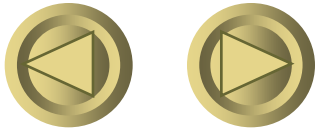
  ANSI SQL, SQL-92 or SQL2, SQL99 or SQL3 extends SQL2 with object-relational features. SQL2003 is the collection of extensions to SQL3.

- **How to query the database**
- **How to make modifications on database**
- **Transactions in SQL**



Why SQL? Or sequel

- **SQL is a very-high-level language.**
  - **Say "what to do" rather than "how to do it."**
  - **Avoid a lot of data-manipulation details needed in procedural languages like C++ or Java.**
- **Database management system figures out "best" way to execute query.**
  - **Called "query optimization."**



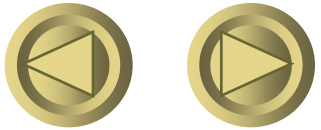
SQL:structured query language

- **Components of language:**

Schema definition, Data retrieval, Data modification, Indexes, Constraints, Views, Triggers, Transactions, authorization,etc

- DDL = data definition language
- DML = data Manipulation Language

- Two forms of usage:
- Interactive SQL (GUI, prompt)
- Embedded SQL (C, Java)

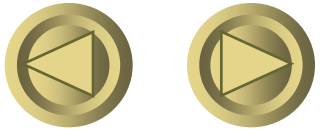


SQL:Structured Query Language

Form

SELECT <desired attributes>
FROM <tuple variables or relation name>
WHERE <conditions>
GROUP BY <attributes>
HAVING <conditions>
ORDER BY < list of attributes>

- **Queries on one relation**
- **Queries on more than one relations**
- **Subqueries and correlated subqueries**
- **Full-relation operations**



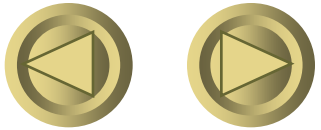
Questions 1:

- **Explain the difference between:**

```
SELECT b
FROM R
WHERE a<10 OR a>=10;
```

and

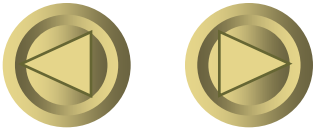
```
SELECT b
FROM R;
```



Question 2: explain the difference between:

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S)
```

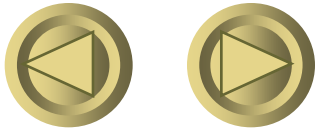


SQL Queries

- **Principal form:**
  - **SELECT desired attributes**
  - **FROM tuple variables — range over relations**
  - **WHERE condition about tuple variables;**

Running example relation schema:

- **Beers(name, manf)**
- **Bars(name, addr, license)**
- **Drinkers(name, addr, phone)**
- **Likes(drinker, beer)**
- **Sells(bar, beer, price)**
- **Frequents(drinker, bar)**



Example: Query on one relation

What beers are made by Anheuser-Busch?

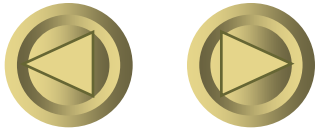
Beers(name, manf)

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

- **Note: single quotes for strings.**

name
Bud
Bud Lite
Michelob

The answer is a relation with a single attribute



Formal Semantics of Single-Relation SQL Query

1. **Start with the relation in the FROM clause.**
2. **Apply (bag) σ, using condition in WHERE clause.**
3. **Apply (extended, bag) π using attributes in SELECT clause.**

Equivalent Operational Semantics

Imagine a *tuple variable* ranging over all tuples of the relation. For each tuple:

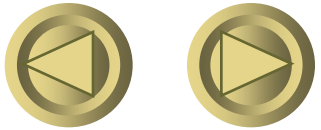
- **Check if it satisfies the WHERE clause.**
- **Print the values of terms in SELECT, if so.**

Star as List of All Attributes

Beers(name, manf)

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch



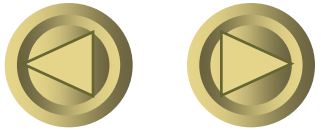
Renaming columns

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute. For example:

Beers(name, manf)

```
SELECT name AS beer
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

beer
Bud
Bud Lite
Michelob



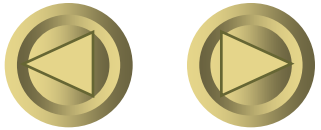
Expressions as Values in Columns

Sells(bar, beer, price)

```
SELECT bar, beer,
    price*6.5 AS priceInRMB
FROM Sells;
```

bar	beer	priceInRMB
Joe's	Bud	19
Sue's	Miller	20
…	…	…

- **Note: no WHERE clause is OK.**

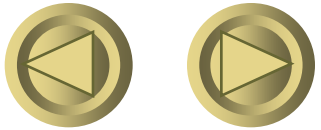


• If you want an answer with a particular string in each row, use that constant as an expression.

Likes(drinker, beer)

```
SELECT drinker,
        'likes Bud' AS whoLikesBud
FROM Likes
WHERE beer = 'Bud';
```

drinker	**whoLikesBud**
Sally	likes Bud
Fred	likes Bud
…	…



Example

- **Find the price Joe's Bar charges for Bud.**

  Sells(bar, beer, price)

  ```
  SELECT price
  FROM Sells
  WHERE bar = 'Joe''s Bar' AND
        beer = 'Bud';
  ```

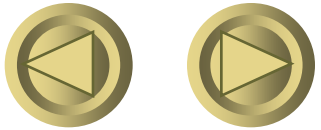
- **Note: two single-quotes in a character string represent one single quote.**
- **Conditions in WHERE clause can use logical operators AND, OR, NOT and parentheses in the usual way.**
- **Remember: SQL is *case insensitive*. Keywords like SELECT or AND can be written upper/lower case as you like.**
  - **Only inside quoted strings does case matter.**

Patterns

- WHERE clauses can have conditions in which a string is compared with a pattern, to see if it matches.

General form:
<Attribute> LIKE <pattern> or
<Attribute> NOT LIKE <pattern>

Pattern is a quoted string with % = “any string”; _ = “any character.”



Pattern Example

- **Find drinkers whose phone has exchange 555.**

  Drinkers(name, addr, phone)

```
SELECT name
FROM Drinkers
WHERE phone LIKE '%555-_ _ _ _';
```

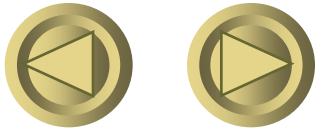

Escape Characters in Like expressions

- **SQL allows to specify any one character we like as the escape character for a single pattern.**
- **Example**

s LIKE ‘x%%x%’ ESCAPE ‘x’

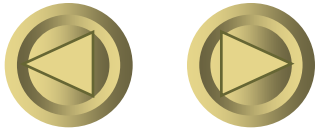
x: escape character in the pattern.

s matches %asd% or %y%;



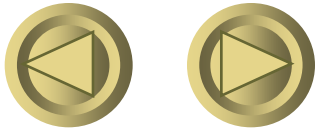
Nulls

- Tuples in SQL relations can have NULL as a value for one or more components.
- Meaning depends on context. Two common cases:
  - *Missing value* : e.g., we know Joe's Bar has some address, but we don't know what it is.
  - *Inapplicable* : e.g., the value of attribute *spouse* for an unmarried person.



Comparing NULL's to Values

- **The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.**
- **Comparing any value (including NULL itself) with NULL yields UNKNOWN.**
- A tuple is in a result iff the WHERE clause is **TRUE** (not FALSE or UNKNOWN).



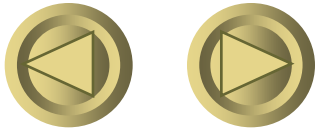
Operation upon on NULL value

- **Operate on a NULL and any value, including another NULL, using an arithmetic operator like * or +, the result is NULL.**

a	b
1	2
null	3
2	null
2	5

Select a, b*6.0 as priceInRMB
From R
Where a >1

a	priceInRMB
2	null
2	30



Question: what is the result?

Where clause:

Where a > 1 AND b< 3

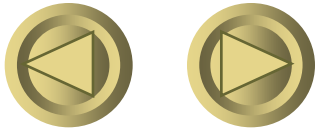
If a=2, a>1 is true

If a=1, a>1 is false

If a is null, a >1 is unknown

Generally,

TRUE AND (FALSE OR NOT(UNKNOWN)) = ?



Three-Valued Logic (See fig6.2)

- **Think of TRUE = 1, FALSE = 0, and UNKNOWN = ½.**
- **AND = MIN; OR = MAX, NOT(*x*) = 1-*x*.**
- **Example:**

TRUE AND (FALSE OR NOT(UNKNOWN)) =

MIN(1, MAX(0, (1 - ½))) =

MIN(1, MAX(0, ½)) =

MIN(1, ½) = ½.

Example

bar	beer	price
Joe's bar	Bud	NULL

```
SELECT bar
FROM Sells
WHERE price < 2.00 OR price >= 2.00;
```

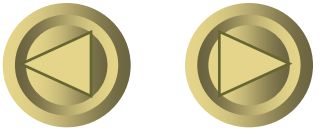
UNKNOWN UNKNOWN

UNKNOWN

- Joe's Bar is not produced.

Reason: 2-Valued Laws != 3-Valued Laws

- **Some common laws, like commutativity of AND, hold in 3-valued logic.**
- **But not others, e.g., the *law of the excluded middle* : *p* OR NOT *p* = TRUE.**
  - **When *p* = UNKNOWN, the left side is MAX(½, (1 – ½)) = ½ != 1.**

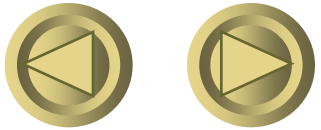


Testing for NULL

- **Use value IS NULL or value IS NOT NULL.**
- **Select * from Sells where price is NULL;**

bar	beer	price
Joe's bar	Bud	NULL

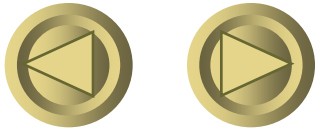
Null is a special value, while unknown is a truth-value, like true or false, is a result of the comparison, or evaluation on a condition.



For example: find an equivalent query

```
Select *
From Sells
Where price <=12 or price >12;

Select  *
From Sells
Where price is not null;
```



Multi-relation Queries

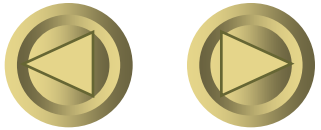
- **Interesting queries often combine data from more than one relation.**
- **List of relations in FROM clause.**
- **Relation-dot-attribute disambiguates attributes from several relations.**

Example: Find the beers that the frequenters of Joe's Bar like.

Likes(drinker, beer)

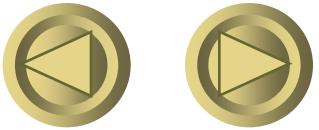
Frequents(drinker, bar)

```
SELECT beer
FROM Frequents, Likes
WHERE bar = 'Joe''s Bar' AND
    Frequents.drinker = Likes.drinker;
```



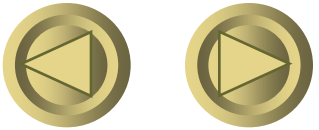
Formal Semantics

- **Almost the same as for single-relation queries:**
  1. **Start with the product of all the relations in the FROM clause.**
  2. **Apply the selection condition from the WHERE clause.**
  3. **Project onto the list of attributes and expressions in the SELECT clause.**

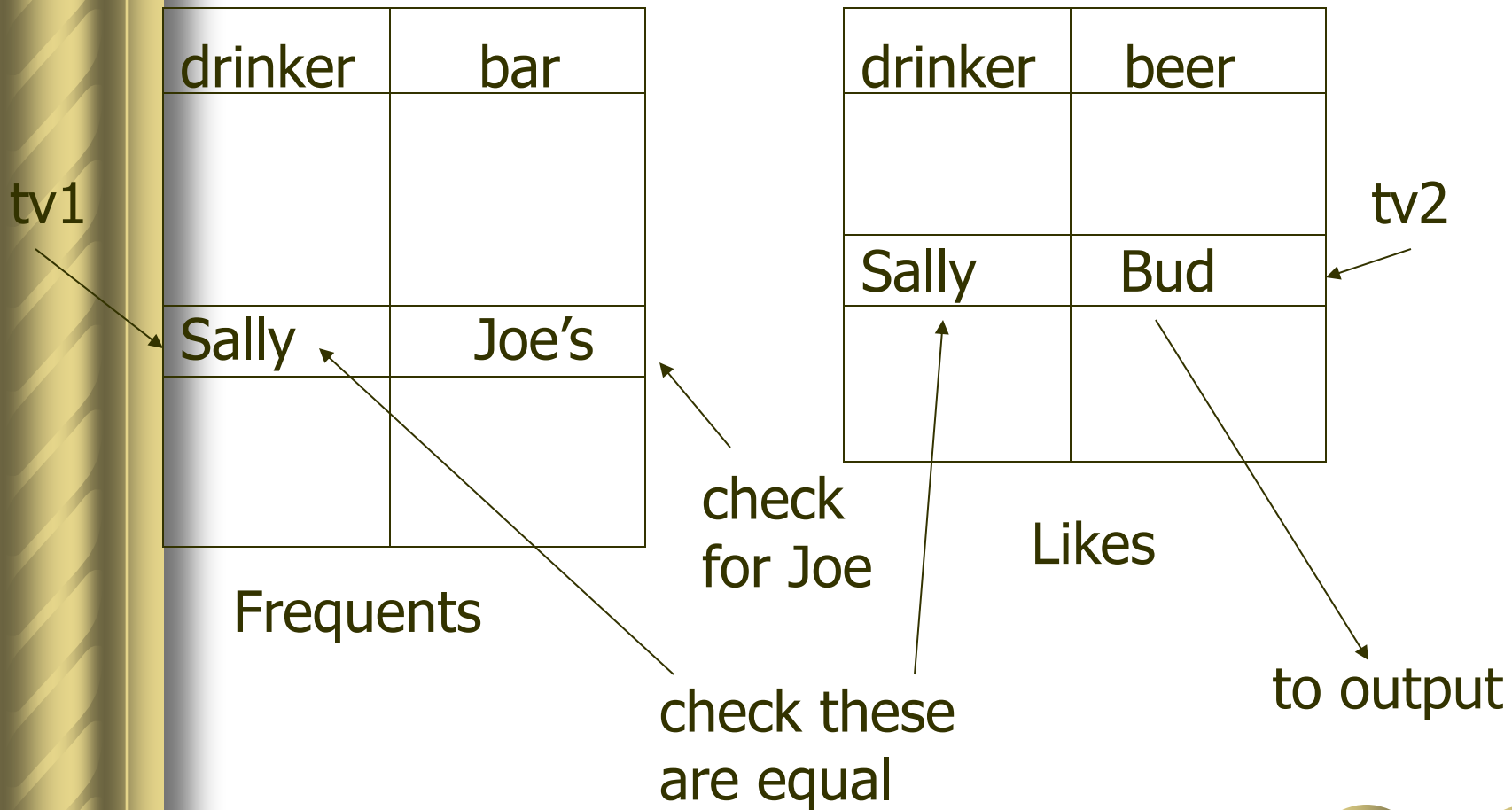


Operational Semantics

- **Imagine one tuple-variable for each relation in the FROM clause.**
  - **These tuple-variables visit each combination of tuples, one from each relation.**
- **If the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause.**

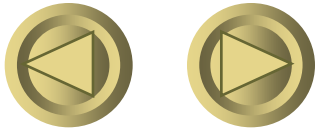


Example


drinker
bar
Sally
Joe's
Frequents
tv1
drinker
beer
Sally
Bud
Likes
tv2
check
for Joe
check these
are equal
to output

Explicit Tuple-Variables

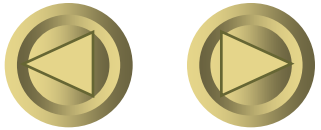
- **Sometimes, a query needs to use two copies of the same relation.**
- **Distinguish copies by following the relation name by the name of a tuple-variable, in the FROM clause.**
- **It's always an option to rename relations this way, even when not essential.**



Example: Self-Join

- **From Beers(name, manf), find all pairs of beers by the same manufacturer.**
  - **Do not produce pairs like (Bud, Bud).**
  - **Produce pairs in alphabetic order, e.g. (Bud, Miller), not (Miller, Bud).**

```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
    b1.name < b2.name;
```



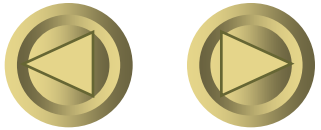
Computer: R intersection (S union T) when T is empty

- **R(a)={100,1}**
- **S(a)={100};**
- **T(a) is empty;**

Select R.a from R,S,T

where R.a=S.a or R.a =T.a;

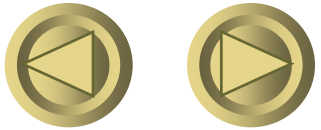
What is the result?



Summary

- **SQL basic queries**
- **Semantics of SQL queries.**

SELECT <desired attributes>
FROM <tuple variables or relation name>
WHERE <conditions>
GROUP BY <attributes>
HAVING <conditions>
ORDER BY < list of attributes>



What is the difference?

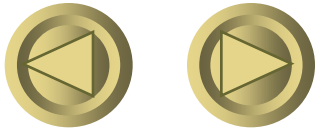
We suppose:

R (a,b)

S (b,c)

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```



IN is a Predicate About R's Tuples

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Two 2's

One loop, over the tuples of R

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) satisfies the condition; 1 is output once.

This Query Pairs Tuples from R, S

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

Double loop, over the tuples of R and S

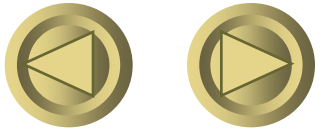
a	b
1	2
3	4

R

b	c
2	5
2	6

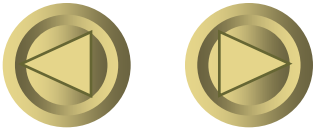
S

(1,2) with (2,5) and (1,2) with (2,6) both satisfy the condition; 1 is output twice.



About the SQLlite

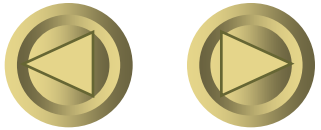
- **SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.**



Classroom exercises

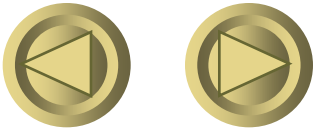
- **Download sqllite and dbdata in the web site: http://www.cs.sjtu.edu.cn/~li-fang/DB.htm**

- **.read mydb.sql**



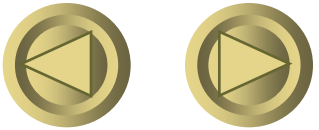
Classroom Exercises: to create a student course database system.

- **create table students(sid int primary key,name char[10],dept char[2],age int default 20);**
- **create table courses (cid int primary key, cname char[10], spring boolean, teacher char[10]);**
- **create table sc (sid int, cid int,semester int,cname varchar[20],grade int);**



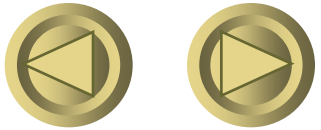
Classroom Exercises:

- **Know all the courses in the spring semester.**
- **Search any ‘data’ courses, such as database, data mining and so on.**
- **Is there a course named “100% success”?**
- **Find those students who have chosen some courses already.**



Know all the courses in the spring semester.

```
Select cid, cname, ‘springOpened’ as spring, teacher
 from courses
 where spring=1;
```

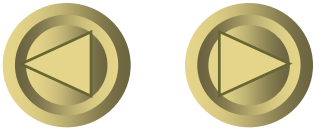


- Search any courses related with ‘data’.

```
select * from courses
where cname like '%data%';
```

- Whether there is a course named “100% success”?

```
select * from courses
where cname like '%X%%' ESCAPE 'X';
```



Find those students who have chosen some courses already.

- **Q1: select name from students,sc where students.sid = sc.sid;**
- **Q2: select name from students where sid in (select sid from sc);**