CS2102
Structured Query Language (SQL)
Part 1

Structured **Q**uery **L**anguage

- Designed by D. Chamberlin & R. Boyce (IBM Research) in 1974
  - Original name: SEQUEL (Structured English QUEry Language)
- SQL is not a general-purpose language but a domain-specific language (DSL)
  - Designed for computations on relations
- Unlike relational algebra which is a procedural language, SQL is a declarative language
  - Focuses on *what* to compute, not on *how* to compute

Using SQL

- **Directly write SQL statements**
  - Command line interface
    - PostgreSQL's psql
      `https://www.postgresql.org/docs/current/static/app-psql.html`
  - Graphical user interface
    - PostgreSQL's pgAdmin
      `https://www.pgadmin.org/`
- **Include SQL in application programs**
  - Statement-Level Interface (SLI)
    - Embedded SQL
    - Dynamic SQL
  - Call-Level Interface (CLI)
    - JDBC (Java DataBase Connectivity)
    - ODBC (Open DataBase Connectivity)

SQL

- Current ANSI/ISO standard for SQL is called SQL:2016
    - Different DBMSs may have minor variations in SQL syntax
- SQL consists of two main parts: DDL & DML
- **Data Definition Language (DDL)**: create/delete/modify schemas
- **Data Manipulation Language (DML)**: ask queries, insert/delete/modify data

Create/Drop Table

```
-- Comments in SQL are preceded
-- by two hyphens
create table Students (
    studentId       integer,
    name            varchar(100),
    birthDate       date
);


/* SQL also supports
C-style comments */
drop table Students;
```

Data Types

- Examples of built-in data types:
  - boolean
  - integer, numeric, real
  - char(50), varchar(50), text
  - date, time, timestamp
- SQL also supports user-defined data types
- The domain of each data type includes the special value null

Null Values

- SQL uses a three-valued logic system: *true*, *false*, & *unknown*

x	y	x AND y	x OR y	NOT x
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	TRUE	FALSE	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	TRUE	
TRUE	FALSE	FALSE	TRUE	
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE	

Null Values (cont.)

- Result of a comparison operation involving *null* value is *unknown*
- Result of an arithmetic operation involving *null* value is *null*
- Examples: Assume that the value of *x* is *null*
  - $x < 100$ evaluates to *unknown*
  - $x = null$ evaluates to *unknown*
  - $x <> null$ evaluates to *unknown*
  - $x + 20$ evaluates to *null*

IS NULL Comparison Predicate

- How to check if a value is equal to *null*?
- Use the IS NULL comparison predicate
- Examples:
  - *x* `IS NULL` evaluates to *true* if *x* is a null value
  - *x* `IS NULL` evaluates to *false* if *x* is a non-null value
  - *x* `IS NOT NULL` evaluates to *false* if *x* is a null value
  - *x* `IS NOT NULL` evaluates to *true* if *x* is a non-null value

IS DISTINCT FROM Comparison Predicate

- How to treat *null* values as ordinary values for comparison?
- Use the IS DISTINCT FROM comparison predicate
- The comparison "x IS DISTINCT FROM y"
  - is equivalent to "x $<>$ y" if both x & y are non-null values
  - evaluates to *false* if both the values are *null*
  - evaluates to *true* if only one of the values is *null*

Constraints in Data Definitions

- **Constraint Types**
  - Not-null constraints
  - Unique constraints
  - Primary key constraints
  - Foreign key constraints
  - General constraints
- **Constraint Specifications**
  - Column constraints
  - Table constraints
  - Assertions (not covered)
- A constraint is violated if it evaluates to *false*

Not-Null Constraints

```
create table Students (
    studentId        integer,
    name             varchar(100) not null,
    birthDate        date
);
```

```
create table Students (
    studentId        integer,
    name             varchar(100),
    birthDate        date,
    check (name is not null)
);
```

Unique Constraints

create table Students (
    studentId   **integer unique**,
    name   **varchar**(100),
    birthDate   **date**
);

The unique constraint is violated if there exists two records $x, y \in \textit{Students}$ where "x.studentId $<>$ y.studentId" evaluates to *false*

studentId	name	birthDate
100	Alice	1999-12-25
200	Bob	2000-04-01
200	Carol	2001-11-28

studentId	name	birthDate
100	Alice	1999-12-25
null	Bob	2000-04-01
null	Carol	2001-11-28

Unique Constraints (cont.)

```
create table Census (
    city        varchar(50),
    state       char(2),
    population  integer,
    unique (city,state)
);
```

city	state	population
New York	NY	8537673
null	CA	3976322
Chicago	IL	2704958
Houston	TX	2303482
null	CA	1406630

The unique constraint is violated if there exists two records $x, y \in \mathit{Census}$ where "(x.city $<>$ y.city) or (x.state $<>$ y.state)" evaluates to *false*

Primary Key Constraints

```
create table Students (
    studentId       integer primary key,
    name            varchar(100),
    birthDate       date
);
```

```
create table Students (
    studentId       integer unique not null,
    name            varchar(100),
    birthDate       date
);
```

Primary Key Constraints (cont.)

```
create table Enrolls (
    sid         integer,
    cid         integer,
    grade       char(2),
    primary key (sid, cid)
);
```

Foreign Key Constraints

```
create table Students (
    studentId       integer,
    name            varchar(100),
    birthDate       date,
    primary key (studentId));
```

```
create table Courses (
    courseId        integer,
    name            varchar(80),
    credits         integer,
    primary key (courseId));
```

```
create table Enrolls (
    sid             integer references Students (studentId),
    cid             integer,
    grade           char(2),
    primary key     (sid, cid),
    foreign key     (cid) references Courses (courseId));
```

Foreign Key Constraints (cont.)

Enrolls (sid, cid, grade) **Students** (studentId, name, birthDate)

Referencing table Referenced table

- Consider a **foreign key constraint** with the subset of attributes A_f in referencing table T_f referring to the subset of attributes A_p in referenced table T_p
- A_p must be declared as **primary key** or **unique** in table T_p
- For each tuple $t_f \in T_f$ with non-null values for all the attributes A_f, the referenced table T_p must contain a tuple t_p where the values of t_p's attributes A_p agree with those of t_f's attributes A_f

General Constraints: Example

```
create table Movies (
    title        varchar(100),
    director     varchar(80),
    releaseYear  integer,
    rating       numeric (3,1),
    primary key (title),
    check (releaseYear > 2010 or rating > 8.0 )
);
```

Database Modifications: Insert

```
create table Students (
    studentId      integer primary key,
    name           varchar(100) not null,
    birthDate      date,
    dept           varchar(20) default 'CS'
);
```

```
insert into Students
values      (12345, 'Alice', '1999-12-25', 'Maths');
```

```
insert into Students (name, studentId)
values      ('Bob', 67890);
```

studentId	name	birthDate	dept
12345	Alice	1999-12-25	Maths
67890	Bob	null	CS

Database Modifications: Delete

```
create table Students (
    studentId     integer primary key,
    name          varchar(100) not null,
    birthDate     date,
    dept          varchar(20) default 'CS'
);
```

```
-- Remove all students
delete from Students;
```

```
-- Remove all students from Maths department
delete from Students
where    dept = 'Maths';
```

Database Modifications: Update

```
create table Accounts (
    accountId       integer primary key,
    name            varchar(100) not null,
    birthDate       date,
    balance         numeric (10,2) default 0.00
);

-- Add 2% interest to all accounts
update Accounts
set     balance = balance * 1.02;

-- Add $500 to account 12345
update    Accounts
set       balance = balance + 500
where     accountId = 12345;
```

Transactions

- A **transaction** consists of one or more update/retrieval operations (i.e., SQL statements)
- Abstraction for representing a logic unit of work
- The **begin** command starts a new transaction
- Each transaction must end with either a **commit** or **rollback** command

```
begin;
update Accounts
set balance = balance + 1000
where accountId = 456;

update Accounts
set balance = balance - 1000
where accountId = 123;
commit;
```

Transactions (cont.)

```
begin;
update Accounts
set balance = balance + 1000
where accountId = 456;

update Accounts
set balance = balance - 1000
where accountId = 123;
commit;
```

vs.

```
update Accounts
set balance = balance + 1000
where accountId = 456;

update Accounts
set balance = balance - 1000
where accountId = 123;
```

Transactions: ACID Properties

- Atomicity: Either all the effects of the transactions are reflected in the database or none are
- Consistency: The execution of a transaction in isolation preserves the consistency of the database
- Isolation: The execution of a transaction is isolated from the effects of other concurrent transaction executions
- Durability: The effects of a committed transaction persists in the database even in the presence of system failures

Checking of Contraints

- By default, constraints are checked immediately at the end of SQL statement execution
  - A violation will cause the statement to be rollbacked
- Constraint checking could also be deferred to the end of transaction execution
  - A violation will cause the transaction to be aborted
- The type of contraint checking could be specified as part of constraint declaration or configure using **set constraints** command

Handling foreign key constraint violations

Enrolls (sid, cid, grade) **Courses** (courseId, name, credits)

Referencing table Referenced table

- Deletion/update of a referenced tuple could violate a foreign key constraint
- Specify action to deal with violation as part of a foreign key constraint declaration:

FOREIGN KEY ... REFERENCES ... ON DELETE/UPDATE action

Handling foreign key constraint violations (cont.)

- NO ACTION: rejects delete/update if it violates constraint (default option)
- RESTRICT: similar to *NO ACTION* except that constraint checking can't be deferred
- CASCADE: propagates delete/update to referencing tuples
- SET DEFAULT: updates foreign keys of referencing tuples to some default value
- SET NULL: updates foreign keys of referencing tuples to *null* value

Handling foreign key constraint violations (cont.)

```
create table Students (
    studentId     integer,
    name          varchar(100),
    birthDate     date,
    primary key (studentId));
```

```
create table Courses (
    courseId      integer,
    name          varchar(80),
    credits       integer,
    primary key (courseId));
```

```
create table Enrolls (
    sid integer,   cid integer,   grade char(2),
    primary key (sid, cid),
    foreign key  (sid) references Students
        on delete cascade
        on update no action,
    foreign key  (cid) references Courses
        on update cascade
        on delete set null);
```

Modifying Schema with ALTER TABLE Command

- Add/remove/modify columns
    - **alter table** Students **alter column** dept **drop default**;
    - **alter table** Students **drop column** dept;
    - **alter table** Students **add column** faculty **varchar**(20);
    - etc.
- Add/remove constraints
- etc.

Simple Queries

- Basic form of SQL query consists of three clauses:

  select [**distinct**] select-list
  from from-list
  [**where** qualification]

- from-list: Specifies list of relations
- qualification: Specifies conditions on relations
- select-list: Specifies columns to be included in output table
- Output relation could contain duplicate records if **distinct** is not used in the select clause

Simple Queries (cont.)

select distinct	$a_1, a_2, \cdots, a_m$
from	$r_1, r_2, \cdots, r_n$
where	c

$$\pi_{a_1,a_2,\cdots,a_m}(\sigma_c(r_1 \times r_2 \times \cdots \times r_n))$$

Simple Queries (cont.)

Find the names of restaurants, the pizzas that they sell and their prices, where the price is under $20

```
select rname, pizza, price
from   Sells
where  price < 20;
```

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname	pizza	price
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Pizza King	Diavola	17

Simple Queries (cont.)

Find the names of restaurants, the pizzas that they sell and their prices, where the price is under $20

```
select rname, pizza, price
from   Sells
where  price < 20;
```

```
select *
from   Sells
where  price < 20;
```

Simple Queries (cont.)

Find all tuples from `Sells` relation such that (1) the price is under $20 and the restaurant name is not “Pizza King” or (2) the restaurant name is “Corleone Corner”.

select ∗ **from** Sells
where ((price < 20) **and** (rname <> ’Pizza King’))
or rname = ’Corleone Corner’;

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma’s Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16

Removing Duplicate Records

Q1: **select** A, C **from** R;

Q2: **select distinct** A, C **from** R;

R

A	B	C
10	1	2
10	7	2
20	3	null
20	9	null
30	3	2
30	5	9

Q1

A	C
10	2
10	2
20	null
20	null
30	2
30	9

Q2

A	C
10	2
20	null
30	2
30	9

Two tuples (a_1, c_1) and (a_2, c_2) in R are considered to be distinct if the following evaluates to *true*:

"(a_1 IS DISTINCT FROM a_2) or (c_1 IS DISTINCT FROM c_2)"

Expressions in SELECT Clause

select item, price $*$ qty **as** cost
from Orders;

Orders

item	price	qty
A	2.50	100
B	4.00	100
C	7.50	100

item	cost
A	250.00
B	400.00
C	750.00

Expressions in SELECT Clause (cont.)

select ’Price of ’ || pizza || ’ is ’ || **round**(price / 1.3) || ’ USD’ **as** menu
from Sells
where rname = ’Corleone Corner’;

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma’s Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

menu
Price of Diavola is 11 USD
Price of Hawaiian is 20 USD
Price of Margherita is 15 USD

Pattern Matching with LIKE Operator

Find customer names ending with “e” that consists of at least four characters

select cname **from** Customers **where** cname **like** ’_ _ _%e’;

Customers

cname	area
Homer	West
Lisa	South
Maggie	East
Moe	Central
Ralph	Central
Willie	North

cname
Maggie
Willie

- The underscore symbol _ matches any single character
- The percent symbol % matches any sequence of 0 or more characters

Set Operations

- Let Q_1 & Q_2 denote SQL queries that output union-compatible relations
- Q_1 **union** Q_2 = $Q_1 \cup Q_2$
- Q_1 **intersect** Q_2 = $Q_1 \cap Q_2$
- Q_1 **except** Q_2 = $Q_1 - Q_2$
- **union**, **intersect**, and **except** eliminate duplicate records
- **union all**, **intersect all**, and **except all** preserves duplicate records

Set Operations (cont.)

Example 1: Find all customer/restaurant names

select cname **from** Customers
union
select rname **from** Restaurants;

Example 2: Find pizzas that contain both cheese and chilli

select pizza **from** Contains **where** ingredient = ’cheese’
intersect
select pizza **from** Contains **where** ingredient = ’chilli’;

Example 3: Find pizzas that contain cheese but not chilli

select pizza **from** Contains **where** ingredient = ’cheese’
except
select pizza **from** Contains **where** ingredient = ’chilli’;

Set Operations (cont.)

Q1: **select** B **from** R **except** **select** B **from** S;
Q2: **select** B **from** R **except all** **select** B **from** S;

R

A	B
10	1
20	1
30	1
40	2
50	3
60	4
70	4

S

A	B
10	1
20	5
30	2
40	2
50	3

Q1

B
4

Q2

B
1
1
4
4

Multi-relation Queries

Find customers and restaurants that are located in the same area

```
select   cname, rname
from     Customers, Restaurants
where    Customers.area = Restaurants.area;
```

```
select   cname, rname
from     Customers as C, Restaurants as R
where    C.area = R.area;
```

Multi-relation Queries (cont.)

Find distinct restaurant pairs (R1,R2) where R1 $<$ R2 and they sell some common pizza

```
select   distinct S1.rname, S2.rname
from     Sells S1, Sells S2
where    S1.rname < S2.rname
and      S1.pizza = S2.pizza;
```

Join Operators in Relational Algebra

- A join operator combines cross-product, selection, and possibly projection operators
- More convenient to use than plain cross-product operator
- Join operators:
  - Natural join, $\bowtie$
  - Inner join (aka join, theta join, or condition join), $\bowtie_c$
  - Left outer join (aka left join) $\rightarrow_c$
  - Right outer join (aka right join) $\leftarrow_c$
  - Full outer join (aka full join) $\leftrightarrow_c$
  - Others: Natural left/right/outer joins

Natural Join: $R \bowtie S$

The natural join of R and S is defined as

$$R \bowtie S = \pi_{\ell}(\sigma_c(R \times S))$$

where
A = common attributes between R & S = $\{a_1, a_2, \cdots, a_n\}$,
c is "$(R.a_1 = S.a_1) \wedge (R.a_2 = S.a_2) \wedge \cdots \wedge (R.a_n = S.a_n)$",
ℓ is the list of attributes in A, followed by the list of attributes in R (excluding those in A) and the list of attributes in S (excluding those in A)

Natural Join: Example

For each restaurant, find its name, area, and the pizzas it sells together with their prices

Restaurants

rname	area
Corleone Corner	North
Gambino Oven	Central
Lorenzo Tavern	Central
Mamma's Place	South
Pizza King	East

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

Restaurants $\bowtie$ Sells

rname	area	pizza	price
Corleone Corner	North	Diavola	24
Corleone Corner	North	Hawaiian	25
Corleone Corner	North	Margherita	19
Gambino Oven	Central	Siciliana	16
Lorenzo Tavern	Central	Funghi	23
Mamma's Place	South	Marinara	22
Pizza King	East	Diavola	17
Pizza King	East	Hawaiian	19

Restaurants $\bowtie$ Sells = $\pi_{Restaurants.rname, area, pizza, price}(\sigma_c(Restaurants \times Sells))$
where c is "Restaurant.rname = Sells.rname"

Natural Join: Example (cont.)

For each restaurant, find its name, area, and the pizzas it sells together with their prices

```
select   R.rname, R.area, S.pizza, S.price
from     Restaurants R, Sells S
where    R.rname = S.rname;
```

```
select   *
from     Restaurants natural join Sells;
```

Natural Join: Example (cont.)

Find distinct names of restaurants that sell some pizza for under $20 that Homer likes

```
select   distinct S.rname
from     Sells S, Likes L
where    S.price < 20
and      L.cname = 'Homer'
and      S.pizza = L.pizza;
```

```
select   distinct S.rname
from     Sells S natural join Likes L
where    S.price < 20
and      L.cname = 'Homer';
```

Inner Join: $R \bowtie_c S$

The inner join of R and S is defined as

$$R \bowtie_c S = \sigma_c(R \times S)$$

Example: Find customer pairs $(C1, C2)$ such that they like some common pizza and $C1 < C2$

Likes

cname	pizza
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Sciliana
Ralph	Diavola

$\pi_{cname,cname2}(Likes \bowtie_c \rho_{Likes2(cname2,pizza2)}(Likes))$
where $c = (pizza = pizza2) \wedge (cname < cname2)$

cname	cname2
Lisa	Maggie
Lisa	Moe
Maggie	Moe

Inner Join: $R \bowtie_c S$ (cont.)

Find customer pairs $(C1, C2)$ such that they like some common pizza and $C1 < C2$

```
select   distinct L1.cname, L2.cname
from     Likes L1, Likes L2
where    L1.cname < L2.cname
and      L1.pizza = L2.pizza;
```

```
select   distinct L1.cname, L2.cname
from     Likes L1 inner join Likes L2
         on (L1.pizza = L2.pizza) and (L1.cname < L2.cname);
```

Left Outer Join: $R \rightarrow_c S$

- $attr(R)$ denote the list of attributes in the schema of R
  - Example: $attr(Sells)$ = rname,pizza,price
- Let $null(R)$ denote a *n*-component tuple of null values, where n = arity of relation R
  - Example: $null(Sells)$ = (null,null,null)

The left outer join of R and S is defined as

$$R \rightarrow_c S = (R \bowtie_c S) \cup ((R \triangleright_c S) \times \{null(S)\})$$

where

$R \triangleright_c S = R - (R \ltimes_c S)$, // left antijoin of R & S

$R \ltimes_c S = \pi_{attr(R)}(R \bowtie_c S)$ // left semijoin of R & S

Left Outerjoin: Example

- Find customers and the pizzas they like; include also customers who don't like any pizza

Customers

cname	area
Homer	West
Lisa	South
Maggie	East
Moe	Central
Ralph	Central
Willie	North

Likes

cname	pizza
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Sciliana
Ralph	Diavola

cname	pizza
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Sciliana
Ralph	Diavola
Willie	null

$$\pi_{Customers.cname,pizza}(Customers \rightarrow_c Likes)$$

where c is "Customers.cname = Likes.cname"

Left Outerjoin: Example (cont.)

Find customers and the pizzas they like; include also customers who don't like any pizza

```
select   C.cname, L.pizza
from     Customers C left outer join Likes L
         on C.cname = L.cname;
```

```
select   C.cname, L.pizza
from     Customers C natural left outer join Likes L;
```

Full Outer Join: $R \leftrightarrow_c S$

The full outer join of R and S is defined as

$$R \ \leftrightarrow_c \ S = (R \rightarrow_c S) \cup (\{null(R)\} \times (S \triangleright_c R))$$

Full Outerjoin: Example

Find customer-restaurant pairs (C,R) where C and R are located in the same area. Include customers that are not co-located with any restaurant, and include restaurants that are not co-located with any customer

Customers

cname	area
Homer	North
Lisa	South
Maggie	East
Moe	Central
Ralph	Central
Willie	North

Restaurants

rname	area
Corleone Corner	West
Gambino Oven	East
Lorenzo Tavern	Central
Mamma's Place	South
Pizza King	East

cname	rname
Homer	null
Lisa	Mamma's Place
Maggie	Gambino Oven
Maggie	Pizza King
Moe	Lorenzo Tavern
Ralph	Lorenzo Tavern
Willie	null
null	Corleone Corner

Full Outerjoin: Example (cont.)

Find customer-restaurant pairs (C,R) where C and R are located in the same area. Include customers that are not co-located with any restaurant, and include restaurants that are not co-located with any customer

```
select  C.cname, R.rname
from    Customers C full outer join Restaurants R
        on C.area = R.area;
```

```
select  C.cname, R.rname
from    Customers C natural full outer join Restaurants R;
```

Summary

- SQL is the standard query language for relational DBMS
- Basic form for querying consists of SELECT, FROM, & WHERE clauses

$$
\begin{array}{ll}
\textbf{select distinct} & a_1, a_2, \cdots a_m \\
\textbf{from} & r_1, r_2, \cdots r_n \\
\textbf{where} & c
\end{array}
$$

$$\pi_{a_1, a_2, \cdots, a_m}(\sigma_c(r_1 \times r_2 \times \cdots \times r_n))$$