CS2102 Lecture 3
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
    - E.g., 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:2019
  - 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;
drop table if exists Students;
drop table if exists Students cascade;
```

Data Types

- Examples of built-in data types:

boolean	false/true (null represents unknown)
integer	signed four-byte integer
float8	double-precision floating point number (8 bytes)
numeric	arbitrary precision floating point number number
numeric(p,s)	maximum total of p digits with maximum of s digits in fractional part
char(n)	fixed-length string consisting of n characters
varchar(n)	variable-length string up to n characters
text	variable-length character string
date	calendar date (year, month, day)
timestamp	date and time

- Other data types: enumerated data type, array, XML, JSON, user-defined data type, etc.
- Reference: `https://www.postgresql.org/docs/current/datatype.html`

Null Values

- Result of a comparison operation involving *null* value is *unknown*
- Result of an arithmetic operation involving *null* value is *null*
- Example: 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*

Null Values (cont.)

- 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	

IS NULL Comparison Predicate

- How to check if a value is equal to *null*?
- Use the IS NULL comparison predicate
  - *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`" is equivalent to "NOT (x IS NULL)"

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*
- "x IS NOT DISTINCT FROM y" is equivalent to "NOT (x IS DISTINCT FROM y)"

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
- A constraint is violated if it evaluates to *false*

Not-Null Constraints

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

The not-null constraint is violated if there exists some record $x \in$ *Students* where "x.name IS NOT NULL" evaluates to *false*

Unique Constraints

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

The unique constraint is violated if there exists two records $x, y \in$ *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 $x, y \in Census$ 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.)

```
create table Rooms (
        level           integer,
        number          integer,
        capacity        integer,
        primary key (level, number)
);
```

```
create table Events (
        eid             integer,
        ename           varchar(100),
        date            date,
        level           integer,
        number          integer,
        primary key (eid),
        foreign key     (level, number)
            references Rooms (level, number)
);
```

Rooms

level	number	capacity
1	1	30
1	2	100
2	1	60
2	2	20
2	3	70

Events

eid	ename	date	level	number
101	CS Seminar	2018-01-02	1	2
102	PhD Defense	2018-03-10	2	3
103	Job Fair	2018-03-14	2	null
104	CS Interviews	2018-06-24	null	null
105	CS Meeting	2018-06-20	null	9

Foreign Key Constraints (cont.)

```
create table Rooms (
        level           integer,
        number          integer,
        capacity        integer,
        primary key (level, number)
);
```

```
create table Events (
        eid             integer,
        ename           varchar(100),
        date            date,
        level           integer,
        number          integer,
        primary key (eid),
        foreign key     (level, number)
            references Rooms (level, number)
            match full
);
```

Rooms

level	number	capacity
1	1	30
1	2	100
2	1	60
2	2	20
2	3	70

Events

eid	ename	date	level	number
101	CS Seminar	2018-01-02	1	2
102	PhD Defense	2018-03-10	2	3
104	CS Interviews	2018-06-24	null	null

Foreign Key Constraints (cont.)

```
create table Rooms (
   level          integer,
   number         integer,
   capacity       integer,
   primary key (level, number),
);

create table Events (
   eid            integer,
   ename          varchar(100),
   date           date,
   level          integer,
   number         integer,
   primary key (eid),
   foreign key (level, number)
      references Rooms (level, number)
);
```

```
create table Rooms (
   level          integer,
   number         integer,
   capacity       integer,
   primary key (level, number),
);

create table Events (
   eid            integer,
   ename          varchar(100),
   date           date,
   level          integer,
   number         integer,
   primary key (eid),
   foreign key (level, number)
      references Rooms
);
```

Foreign Key Constraints (cont.)

```
create table Rooms (
    rid          integer,
    level        integer,
    number       integer,
    capacity     integer,
    primary key (rid),
    unique (level, number)
);
create table Events (
    eid          integer,
    ename        varchar(100),
    date         date,
    level        integer,
    number       integer,
    primary key (eid),
    foreign key (level, number) references Rooms (level, number)
);
```

Check Constraints

```
create table Lectures (
    cname       char(5),
    pname       varchar(50) not null,
    day         smallint
                check (day in (1,2,3,4,5)),
    hour        smallint
                check ((hour >= 8) and (hour <= 17)),
    primary key (cname,day,hour),
    unique (pname,day,hour)
);
```

Check Constraints (cont.)

```
create table Lectures (
    cname       char(5),
    pname       varchar(50) not null,
    day         smallint,
    hour        smallint,
    primary key (cname,day,hour),
    unique (pname,day,hour),
    check (
      (day in (1,2,3,4,5) and hour >= 8 and hour <= 17)
      or
      (day = 6 and hour in (9,10,11))
    )
);
```

Constraint Names

```
create table Lectures (
    cname       char(5),
    pname       varchar(50)
                constraint lectures_pname check (pname is not null),
    day         smallint
                constraint lectures_day check (day in (1,2,3,4,5)),
    hour        smallint
                constraint lectures_hour check (hour >= 8 and hour <= 17),
    constraint lectures_pri_key primary key (cname,day,hour),
    constraint lectures_cand_key unique (pname,day,hour)
);
```

Assertions

- SQL's **create assertion** statement supports the specification of general constraints
  - Example: Every course is enrolled by at least one student
- However, the create assertion statement is not implemented in many DBMSs
- Instead, general constraints are enforced using **triggers**

Database Modifications: Insert

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

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

insert into Students (name, studentId)
values      ('Bob', 67890), ('Carol', 11122);
```

studentId	name	birthDate	dept
12345	Alice	1999-12-25	Maths
67890	Bob	null	null
11122	Carol	null	null

Database Modifications: Insert (cont.)

```
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), ('Carol', 11122);
```

studentId	name	birthDate	dept
12345	Alice	1999-12-25	Maths
67890	Bob	null	CS
11122	Carol	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 & change name to 'Alice'
update  Accounts
set     balance = balance + 500,
        name = 'Alice'
where   accountId = 12345;
```

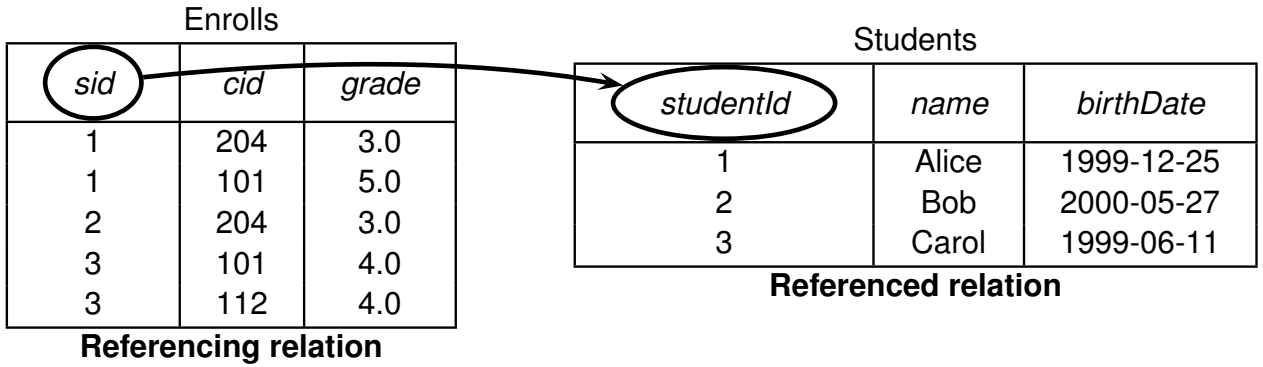
How to retrieve all the records from a table?

select * **from** Customers;

Customers

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

Foreign Key Constraint Violations



- 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
- Default action is “NO ACTION”
  - The delete/update statement that causes violation will be rejected

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

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 cascade,
    foreign key  (cid) references Courses);
```

Foreign Key Constraint Violations (cont.)

foreign key (sid) **references** Students **on delete cascade**

Students

studentId	*name*	*birthDate*
1	Alice	1999-12-25
2	Bob	2000-05-27
3	Carol	1999-06-11

Enrolls

sid	*cid*	*grade*
1	204	3.0
1	101	5.0
2	204	3.0
3	101	4.0
3	112	4.0

```
delete from
Students
where studentId = 1;
```

Students

studentId	*name*	*birthDate*
2	Bob	2000-05-27
3	Carol	1999-06-11

Enrolls

sid	*cid*	*grade*
2	204	3.0
3	101	4.0
3	112	4.0

Foreign Key Constraint Violations (cont.)

foreign key (sid) **references** Students **on update cascade**

Students

studentId	*name*	*birthDate*
1	Alice	1999-12-25
2	Bob	2000-05-27
3	Carol	1999-06-11

Enrolls

sid	*cid*	*grade*
1	204	3.0
1	101	5.0
2	204	3.0
3	101	4.0
3	112	4.0

update Students
set studentId = 4
where studentId = 1;

Students

studentId	*name*	*birthDate*
4	Alice	1999-12-25
2	Bob	2000-05-27
3	Carol	1999-06-11

Enrolls

sid	*cid*	*grade*
4	204	3.0
4	101	5.0
2	204	3.0
3	101	4.0
3	112	4.0

Foreign Key Constraint Violations (cont.)

```
create table  Employees (
        eid             integer primary key,
        ename           varchar(100),
        managerId       integer default 1,
        foreign key     (managerId) references Employees
                        on delete set default
);
```

Employees

eid	*ename*	*managerId*
1	Alice	null
2	Bob	1
3	Carol	2
4	Dave	2

```
delete from
Employees
where eid = 2;
```

Employees

eId	*ename*	*managerId*
1	Alice	null
3	Carol	1
4	Dave	1

Transactions

- A **transaction** consists of one or more update/retrieval operations (i.e., SQL statements)
- Abstraction for representing a logical 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 = 2;

update Accounts
set balance = balance - 1000
where accountId = 1;
commit;
```

Transactions (cont.)

```
begin;
update Accounts
set balance = balance + 1000
where accountId = 2;

update Accounts
set balance = balance - 1000
where accountId = 1;
commit;
```

vs.

```
update Accounts
set balance = balance + 1000
where accountId = 2;

update Accounts
set balance = balance - 1000
where accountId = 1;
```

Transactions: ACID Properties

- Atomicity: Either all the effects of a transaction 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

Deferrable Constraints

- By default, constraints are checked immediately at the end of SQL statement execution
  - A violation will cause the statement to be rollbacked

```
create table  Employees (
        eid             integer primary key,
        ename           varchar(100),
        managerId       integer,
        foreign key     (managerId) references Employees
);

insert into Employees values (1, 'Alice', null), (2, 'Bob', 1), (3, 'Carol', 2);

delete from Employees where eid = 2;
update Employees set managerId = 1 where eid = 3;
```

Deferrable Constraints (cont.)

- The checking could also be deferred for some constraints to the end of transaction execution
  - A violation will cause the transaction to be aborted
  - Deferrable constraints:
    - unique
    - primary key
    - foreign key
- Deferrable constraints are specified using
  - either **deferrable initially deferred**
  - or **deferrable initially immediate**
- The checking of deferrable constraints can be changed using **set constraints** statement

Deferrable Constraints (cont.)

```
create table  Employees (
        eid             integer primary key,
        ename           varchar(100),
        managerId       integer,
        foreign key     (managerId) references Employees
                        deferrable initially deferred
);

insert into Employees values (1, 'Alice', null), (2, 'Bob', 1), (3, 'Carol', 2);

begin;
delete from Employees where eid = 2;
update Employees set managerId = 1 where eid = 3;
commit;
```

Deferrable Constraints (cont.)

```
create table  Employees (
        eid             integer primary key,
        ename           varchar(100),
        managerId       integer,
        constraint employees_fkey  foreign key (managerId)  references Employees
                        deferrable initially immediate
);

insert into Employees values (1, 'Alice', null), (2, 'Bob', 1), (3, 'Carol', 2);

begin;
set constraints employees_fkey deferred;
delete from Employees where eid = 2;
update Employees set managerId = 1 where eid = 3;
commit;
```

Modifying Schema

- 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
  - **alter table** Students **add constraint** fk_grade **foreign key** (grade) **references** Grades;
  - etc.
- etc.
- Reference:

  `https://www.postgresql.org/docs/current/sql-altertable.html`

Summary

- SQL is the standard query language for relational DBMS
- Column/Table Constraints
  - not null
  - unique
  - primary key
  - foreign key
  - check
- Transactions & deferrable constraints
- SQL Reference:
  `https://www.postgresql.org/docs/current/index.html`