CS315: Database Systems
Structured Query Language (SQL)

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

Computer Science and Engineering,
Indian Institute of Technology, Kanpur
http://web.cse.iitk.ac.in/~cs315/

2nd semester, 2019-20

Tue, Wed 12:00-13:15

Structured Query Language (SQL)

- SQL is a *querying* language for relational databases

Structured Query Language (SQL)

- SQL is a *querying* language for relational databases
- Is a data manipulation language (DML)
  - Can access and manipulate data stored as a particular data model

Structured Query Language (SQL)

- SQL is a *querying* language for relational databases
- Is a data manipulation language (DML)
  - Can access and manipulate data stored as a particular data model
- *Declarative language*
  - Specifies what to do, but not how to do

Structured Query Language (SQL)

- SQL is a *querying* language for relational databases
- Is a data manipulation language (DML)
  - Can access and manipulate data stored as a particular data model
- *Declarative language*
  - Specifies what to do, but not how to do
- Is also a data definition language (DDL)
  - Defines database relations and schemas

Structured Query Language (SQL)

- SQL is a *querying* language for relational databases
- Is a data manipulation language (DML)
  - Can access and manipulate data stored as a particular data model
- *Declarative language*
  - Specifies what to do, but not how to do
- Is also a data definition language (DDL)
  - Defines database relations and schemas
- SQL has evolved widely after its first inception
  - Supports lots of extra operations that are non-standard

Example Schema

- course (code, title, *ctype*, webpage)
- coursetype (ctype, *dept*)
- faculty (fid, name, *dept*, designation)
- department (deptid, name)
- semester (yr, half)
- offering (*coursecode, yr, half, instructor*)
- student (roll, name, *dept*, cpi)
- program (*roll*, ptype)
- registration (*coursecode, roll, yr, half, gradecode*)
- grade (gradecode, value)

Creating Relation Schemas

- create table: create table
  $r(A_1\ D_1\ C_1, \ldots, A_n\ D_n\ C_n, (IC_1), \ldots, (IC_k))$
  - r is the name of the relation
  - Each A_i is an attribute name whose data type or domain is specified by D_i
  - C_i specifies constraints or settings (if any)
  - IC_j represents integrity constraints (if any)
- Example

```
create table faculty (
    fid integer primary key,
    name varchar(50) not null,
    dept integer,
    designation varchar(3)
);
```

Data Types in SQL

- *char(n)*: fixed-length character string
- *varchar(n)*: variable-length character string, up to *n*
- *integer* or *int*: integer
- *smallint*: short integer
- *numeric(n,d)*: floating-point number with a total of *n* digits of which *d* is after the decimal point
- *real*: single-precision floating-point number
- *double precision*: double-precision floating-point number
- *float(n)*: floating-point number with at least *n* digits

Data Types in SQL

- *char(n)*: fixed-length character string
- *varchar(n)*: variable-length character string, up to *n*
- *integer* or *int*: integer
- *smallint*: short integer
- *numeric(n,d)*: floating-point number with a total of *n* digits of which *d* is after the decimal point
- *real*: single-precision floating-point number
- *double precision*: double-precision floating-point number
- *float(n)*: floating-point number with at least *n* digits
- *date*: yyyy-mm-dd format
- *time*: hh:mm:ss format
- *time(i)*: hh:mm:ss:i...i format with additional *i* digits for fraction of a second
- *timestamp*: both date and time
- *interval*: relative value in either year-month or day-time format

Other Data Types

- User-defined data type

```
create type cpi as numeric(3,1);
```

- Large objects such as images, videos, strings can be stored as
  - blob: binary large object
  - clob: character large object
  - A pointer to the object is stored in the relation, and not the object itself

Other Data Types

- User-defined data type

```
create type cpi as numeric(3,1);
```

- Large objects such as images, videos, strings can be stored as
  - blob: binary large object
  - clob: character large object
  - A pointer to the object is stored in the relation, and not the object itself
- User-defined domain

```
create domain name as varchar(50) not null;
```

Constraints

- Can be specified for each attribute as well as separately
  - *not null*: the attribute cannot be null
    - Requires some value while inserting as otherwise null is the default
  - *primary key* $(A_i, \ldots, A_j)$: automatically ensures not null
  - *default n*: defaults to *n* if no value is specified
  - *unique*: specifies that this is a candidate key
  - *foreign key*: specifies as a foreign key and the relation it refers to
  - *check P*: predicate *P* must be satisfied

```
create table faculty (
   fid integer,
   name varchar(50) not null,
   dept integer,
   designation varchar(3) default ''AP'',
   primary key fid,
   foreign key (dept) references department(dept),
   check (fid >= 0)
);
```

Deleting or Modifying a Relation Schema

- drop table: drop table *r* deletes the table from the database
  - Must satisfy other constraints already applied
- Example

```
drop table faculty;
```

Deleting or Modifying a Relation Schema

- drop table: drop table *r* deletes the table from the database
  - Must satisfy other constraints already applied
- Example

```
drop table faculty;
```

- alter table: alter table *r* add *A D C*
  - Adds attribute *A* with data type *D* at the end
  - *C* specifies constraints on *A* (if any)
  - Must satisfy other constraints already applied
- alter table: alter table *r* drop *A*
  - Deletes attribute *A* from all tuples
  - Must satisfy other constraints already applied
- Example

```
alter table faculty add room varchar(10);
alter table course drop webpage;
```

Basic Query Structure

- SQL is based on *relational algebra*
- A *basic* SQL query is of the form
  select $A_1, \ldots, A_n$
  from $r_1, \ldots, r_m$
  where P
- Each r_i is a relation
- Each A_j is an attribute from one of $r_1, \ldots, r_m$
- P is a predicate involving attributes and constants
- where can be left out, which then means *true*
- Result is a relation with the schema $(A_1, \ldots, A_n)$

Basic Query Structure

- SQL is based on *relational algebra*
- A *basic* SQL query is of the form
  select $A_1, \ldots, A_n$
  from $r_1, \ldots, r_m$
  where P
- Each r_i is a relation
- Each A_j is an attribute from one of $r_1, \ldots, r_m$
- P is a predicate involving attributes and constants
- where can be left out, which then means *true*
- Result is a relation with the schema $(A_1, \ldots, A_n)$
- Is equivalent to the relational algebra query

Basic Query Structure

- SQL is based on *relational algebra*
- A *basic* SQL query is of the form
  select $A_1, \ldots, A_n$
  from $r_1, \ldots, r_m$
  where P
- Each r_i is a relation
- Each A_j is an attribute from one of $r_1, \ldots, r_m$
- P is a predicate involving attributes and constants
- where can be left out, which then means *true*
- Result is a relation with the schema $(A_1, \ldots, A_n)$
- Is equivalent to the relational algebra query
  $\Pi_{A_1,\ldots,A_n}(\sigma_P(r_1 \times \cdots \times r_m))$

Multisets

- SQL relations are multisets or bags of tuples and not sets
- Consequently, there may be two identical tuples
- This is the biggest distinction with relational algebra

Multisets

- SQL relations are multisets or bags of tuples and not sets
- Consequently, there may be two identical tuples
- This is the biggest distinction with relational algebra
- The set behavior can be enforced by the keyword unique
- In a query, keyword distinct achieves the same effect
- Opposite is keyword all, which is *default*

Select

- Lists attributes in the final output

Select

- Lists attributes in the final output
- Example: Find codes of courses offered in 2018

Select

- Lists attributes in the final output
- Example: Find codes of courses offered in 2018

```
select coursecode
from offering
where yr = 2018;
```

- Case-insensitive
- select * chooses all attributes
- To eliminate duplicates, use select distinct . . .
- Otherwise, by default is select all . . .

Select

- Lists attributes in the final output
- Example: Find codes of courses offered in 2018

```
select coursecode
from offering
where yr = 2018;
```

- Case-insensitive
- select * chooses all attributes
- To eliminate duplicates, use select distinct ...
- Otherwise, by default is select all ...
- Can contain arithmetic expressions

```
select coursecode, yr - 1959
from offering
where yr = 2018;
```

From

- Lists relations from where attributes will be listed
- Corresponds to Cartesian product of the relations

From

- Lists relations from where attributes will be listed
- Corresponds to Cartesian product of the relations
- Example: Find title of courses offered in 2018

From

- Lists relations from where attributes will be listed
- Corresponds to Cartesian product of the relations
- Example: Find title of courses offered in 2018

```
select title
from course, offering
where course.code = offering.coursecode and yr = 2018;
```

From

- Lists relations from where attributes will be listed
- Corresponds to Cartesian product of the relations
- Example: Find title of courses offered in 2018

```
select title
from course, offering
where course.code = offering.coursecode and yr = 2018;
```

- When two relations contain attributes of the same name, qualification is needed to remove ambiguity
- Example: Find name of students in B.Tech. program

From

- Lists relations from where attributes will be listed
- Corresponds to Cartesian product of the relations
- Example: Find title of courses offered in 2018

```
select title
from course, offering
where course.code = offering.coursecode and yr = 2018;
```

- When two relations contain attributes of the same name, qualification is needed to remove ambiguity
- Example: Find name of students in B.Tech. program

```
select student.name
from student, program
where student.roll = program.roll and program.ptype =
    ''B.Tech.'';
```

Where

- Specifies conditions that the result tuples must satisfy

Where

- Specifies conditions that the result tuples must satisfy
- Example

```
select coursecode
from offering
where yr = 2018;
```

Where

- Specifies conditions that the result tuples must satisfy
- Example

```
select coursecode
from offering
where yr = 2018;
```

- May use and, or and not to connect predicates

```
select coursecode
from offering
where yr = 2018 and instructor = 10;
```

Where

- Specifies conditions that the result tuples must satisfy
- Example

```
select coursecode
from offering
where yr = 2018;
```

- May use and, or and not to connect predicates

```
select coursecode
from offering
where yr = 2018 and instructor = 10;
```

- Unused clause is equivalent to where true

```
select coursecode
from offering;
```

Where

- Specifies conditions that the result tuples must satisfy
- Example

```
select coursecode
from offering
where yr = 2018;
```

- May use and, or and not to connect predicates

```
select coursecode
from offering
where yr = 2018 and instructor = 10;
```

- Unused clause is equivalent to where true

```
select coursecode
from offering;
```

- SQL allows between operator (includes both)

```
select coursecode
from offering
where yr between 2016 and 2018;
```

Rename Operation

- SQL allows renaming of relations and attributes to remove ambiguity
- Keyword as is used
- Example

```
select student.roll as rollnumber
from student, program
where student.roll = program.roll and program.ptype =
    ''B.Tech.'';
```

Rename Operation

- SQL allows renaming of relations and attributes to remove ambiguity
- Keyword as is used
- Example

```
select student.roll as rollnumber
from student, program
where student.roll = program.roll and program.ptype =
    ''B.Tech.'';
```

- Renaming is necessary when the same relation needs to be used twice
- Example: Find names of students whose cpi is greater than that of "ABC"

Rename Operation

- SQL allows renaming of relations and attributes to remove ambiguity
- Keyword as is used
- Example

```
select student.roll as rollnumber
from student, program
where student.roll = program.roll and program.ptype =
    ''B.Tech.'';
```

- Renaming is necessary when the same relation needs to be used twice
- Example: Find names of students whose cpi is greater than that of "ABC"

```
select T.name
from student as T, student as S
where T.cpi > S.cpi and S.name = ''ABC'';
```

- as can be omitted by simply stating student T

String Operations

- Supports string matching in addition to equality of two strings
- Uses like to match patterns specified using special characters
  - _: matches any character
  - %: matches any substring

String Operations

- Supports string matching in addition to equality of two strings
- Uses like to match patterns specified using special characters
  - _: matches any character
  - %: matches any substring
- Example: Find all departments having “Engineering” in its name

String Operations

- Supports string matching in addition to equality of two strings
- Uses like to match patterns specified using special characters
  - _: matches any character
  - %: matches any substring
- Example: Find all departments having “Engineering” in its name

```
select *
from department
where name like ''%Engineering%'';
```

String Operations

- Supports string matching in addition to equality of two strings
- Uses like to match patterns specified using special characters
  - _: matches any character
  - %: matches any substring
- Example: Find all departments having “Engineering” in its name

```
select *
from department
where name like ''%Engineering%'';
```

- Example: Find departments with the name “?E”

String Operations

- Supports string matching in addition to equality of two strings
- Uses like to match patterns specified using special characters
  - _: matches any character
  - %: matches any substring
- Example: Find all departments having “Engineering” in its name

```
select *
from department
where name like ''%Engineering%'';
```

- Example: Find departments with the name “?E”

```
select *
from department
where name like ''_E'';
```

String Operations

- Supports string matching in addition to equality of two strings
- Uses like to match patterns specified using special characters
  - _: matches any character
  - %: matches any substring
- Example: Find all departments having “Engineering” in its name

```
select *
from department
where name like ''%Engineering%'';
```

- Example: Find departments with the name “?E”

```
select *
from department
where name like ''_E'';
```

- Example: Find the department whose name is “_E”

String Operations

- Supports string matching in addition to equality of two strings
- Uses like to match patterns specified using special characters
  - _: matches any character
  - %: matches any substring
- Example: Find all departments having “Engineering” in its name

```
select *
from department
where name like ''%Engineering%'';
```

- Example: Find departments with the name “?E”

```
select *
from department
where name like ''_E'';
```

- Example: Find the department whose name is “_E”

```
select *
from department
where name = ''_E'';
```

Ordering of Tuples

- Tuples in the final relation can be ordered using order by

Ordering of Tuples

- Tuples in the final relation can be ordered using order by
  - For display purposes only and has no actual effect
- Example: Order departments by name

Ordering of Tuples

- Tuples in the final relation can be ordered using order by
  - For display purposes only and has no actual effect
- Example: Order departments by name

```
select *
from department
order by name;
```

- Use desc to obtain tuples in descending order

```
select *
from department
order by name desc;
```

Ordering of Tuples

- Tuples in the final relation can be ordered using order by
  - For display purposes only and has no actual effect
- Example: Order departments by name

```
select *
from department
order by name;
```

- Use desc to obtain tuples in descending order

```
select *
from department
order by name desc;
```

- Default is ascending order (asc)

Set Operations

- Operators union, intersect and except correspond to $\cup$, $\cap$, –
- *Eliminates* duplicates
- For multiset operations, i.e., to retain duplicates, use all after the operations
- Example: Find names of all faculty members and students

Set Operations

- Operators union, intersect and except correspond to $\cup$, $\cap$, –
- *Eliminates* duplicates
- For multiset operations, i.e., to retain duplicates, use all after the operations
- Example: Find names of all faculty members and students

```
(select name from faculty)
union
(select name from student);
```

- Example: Find names of faculty members not found in students

Set Operations

- Operators union, intersect and except correspond to $\cup$, $\cap$, –
- *Eliminates* duplicates
- For multiset operations, i.e., to retain duplicates, use all after the operations
- Example: Find names of all faculty members and students

```
(select name from faculty)
union
(select name from student);
```

- Example: Find names of faculty members not found in students

```
(select name from faculty)
except
(select name from student);
```

Aggregate Functions

- Five operations that work on multisets: avg, min, max, sum, count
- Example: Find the average cpi of students

Aggregate Functions

- Five operations that work on multisets: avg, min, max, sum, count
- Example: Find the average cpi of students

```
select avg(cpi)
from student;
```

- For set operations, use distinct
- Example: Find the total number of students

Aggregate Functions

- Five operations that work on multisets: avg, min, max, sum, count
- Example: Find the average cpi of students

```
select avg(cpi)
from student;
```

- For set operations, use distinct
- Example: Find the total number of students

```
select count(distinct roll)
from student;
```

Grouping

- To apply aggregate operations on separate groups, use group by
- The aggregate operator is applied on *each* group separately
- Example: Find the number of students in each department

Grouping

- To apply aggregate operations on separate groups, use group by
- The aggregate operator is applied on *each* group separately
- Example: Find the number of students in each department

```
select dept, count(distinct roll)
from student
group by dept;
```

Grouping

- To apply aggregate operations on separate groups, use group by
- The aggregate operator is applied on *each* group separately
- Example: Find the number of students in each department

```
select dept, count(distinct roll)
from student
group by dept;
```

- Attributes in select clause outside of aggregate functions *must* appear in group by list

Qualifying Groups

- In order to select certain groups, use having clause
- Only those groups satisfying having clause appear in the result
- Example: Find average grade in each course where number of students is at least 5

Qualifying Groups

- In order to select certain groups, use having clause
- Only those groups satisfying having clause appear in the result
- Example: Find average grade in each course where number of students is at least 5

```
select coursecode, avg(grade)
from registration
group by coursecode
having count(roll) >= 5;
```

- The predicate in having is applied *after* forming groups whereas the predicate in where is applied *before* doing so

Qualifying Groups

- In order to select certain groups, use having clause
- Only those groups satisfying having clause appear in the result
- Example: Find average grade in each course where number of students is at least 5

```
select coursecode, avg(grade)
from registration
group by coursecode
having count(roll) >= 5;
```

- The predicate in having is applied *after* forming groups whereas the predicate in where is applied *before* doing so
- Example: Find average grade in each course of type 4 where number of students is at least 5

Qualifying Groups

- In order to select certain groups, use having clause
- Only those groups satisfying having clause appear in the result
- Example: Find average grade in each course where number of students is at least 5

```
select coursecode, avg(grade)
from registration
group by coursecode
having count(roll) >= 5;
```

- The predicate in having is applied *after* forming groups whereas the predicate in where is applied *before* doing so
- Example: Find average grade in each course of type 4 where number of students is at least 5

```
select coursecode, avg(grade)
from registration, course
where registration.coursecode = course.code and ctype = 4
group by coursecode
having count(roll) >= 5;
```

Null

- *null* signifies missing or unknown value
- The predicates is null and is not null can be used to check for null values
- Example: find courses that do not have a webpage

Null

- *null* signifies missing or unknown value
- The predicates is null and is not null can be used to check for null values
- Example: find courses that do not have a webpage

```
select code
from course
where webpage is null;
```

Null

- *null* signifies missing or unknown value
- The predicates is null and is not null can be used to check for null values
- Example: find courses that do not have a webpage

```
select code
from course
where webpage is null;
```

- Result of expressions involving null evaluate to null
- Comparison with null returns *unknown*
- Uses same three-valued logic as relational algebra
- Aggregate functions ignore null
  - count(*) does *not* ignore nulls

Nested Subqueries

- A query that occurs in the where or from clause of another query is called a subquery
- Entire query is called outer query while the subquery is called inner query or nested query
- Used in tests for set membership, set cardinality, set comparisons

Set Membership

- Keyword in is used for set membership tests
- Example: Find faculty members who have not offered any course

Set Membership

- Keyword in is used for set membership tests
- Example: Find faculty members who have not offered any course

```
select *
from faculty
where fid not in (
  select instructor
  from offering );
```

Scoping of Attributes

- It is always a good practice to qualify attributes
- It is better to rename relations if it is used in both the outer and the inner queries

Scoping of Attributes

- It is always a good practice to qualify attributes
- It is better to rename relations if it is used in both the outer and the inner queries
- An unqualified attribute refers to the *innermost* query

Scoping of Attributes

- It is always a good practice to qualify attributes
- It is better to rename relations if it is used in both the outer and the inner queries
- An unqualified attribute refers to the *innermost* query
- When a nested query refers to an attribute in the outer query, it is called a correlated query
- Example: Find names of all students who have taken course with an instructor with the same name

Scoping of Attributes

- It is always a good practice to qualify attributes
- It is better to rename relations if it is used in both the outer and the inner queries
- An unqualified attribute refers to the *innermost* query
- When a nested query refers to an attribute in the outer query, it is called a correlated query
- Example: Find names of all students who have taken course with an instructor with the same name

```
select student.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
  select instructor
  from offering
  where offering.coursecode = R.coursecode );
```

- Inner query is evaluated for *each tuple* in the outer query

Correlated Queries

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O
   where O.coursecode = R.coursecode );
```

student	
roll	name
11	AB
12	CD
13	EF

faculty	
fid	name
101	AB
102	EF
103	GH

registration	
coursecode	roll
1	11
2	12
3	13

Correlated Queries

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O
   where O.coursecode = R.coursecode );
```

student	
roll	name
11	AB
12	CD
13	EF

faculty	
fid	name
101	AB
102	EF
103	GH

registration	
coursecode	roll
1	11
2	12
3	13

(R.roll = S.roll and S.name = F.name)($R \times S \times F$)

R.coursecode	R.roll	S.roll	S.name	F.fid	F.name
1	11	11	AB	101	AB
3	13	13	EF	102	EF

Evaluation Per Tuple

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O
   where O.coursecode = R.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

offering	
O.coursecode	instructor
1	101
2	102
3	103

- $\langle 1, 11, AB, 101 \rangle$

Evaluation Per Tuple

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O
   where O.coursecode = R.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

offering	
O.coursecode	instructor
1	101
2	102
3	103

- $\langle 1, 11, AB, 101 \rangle$
  - With R.coursecode $= 1$, offering chooses only instructor $\{101\}$

Evaluation Per Tuple

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O
   where O.coursecode = R.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

offering	
O.coursecode	instructor
1	101
2	102
3	103

- $\langle 1, 11, AB, 101 \rangle$
  - With R.coursecode $= 1$, offering chooses only instructor $\{101\}$
  - Now, since *fid* $= 101 \in \{101\}$, tuple is returned

Evaluation Per Tuple

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O
   where O.coursecode = R.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

offering	
O.coursecode	instructor
1	101
2	102
3	103

- $\langle 1, 11, AB, 101 \rangle$
  - With R.coursecode $= 1$, offering chooses only instructor $\{101\}$
  - Now, since *fid* $= 101 \in \{101\}$, tuple is returned
- $\langle 3, 13, EF, 102 \rangle$

Evaluation Per Tuple

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O
   where O.coursecode = R.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

offering	
O.coursecode	instructor
1	101
2	102
3	103

- $\langle 1, 11, AB, 101 \rangle$
  - With R.coursecode $=$ 1, offering chooses only instructor $\{101\}$
  - Now, since *fid* $= 101 \in \{101\}$, tuple is returned
- $\langle 3, 13, EF, 102 \rangle$
  - With R.coursecode $=$ 3, offering chooses only instructor $\{103\}$

Evaluation Per Tuple

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O
   where O.coursecode = R.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

offering	
O.coursecode	instructor
1	101
2	102
3	103

- $\langle 1, 11, AB, 101 \rangle$
  - With R.coursecode $= 1$, offering chooses only instructor $\{101\}$
  - Now, since $fid = 101 \in \{101\}$, tuple is returned
- $\langle 3, 13, EF, 102 \rangle$
  - With R.coursecode $= 3$, offering chooses only instructor $\{103\}$
  - Now, since $fid = 102 \notin \{103\}$, tuple is not returned

Non-Correlated Query

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O, registration as G
   where O.coursecode = G.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

Inner query
O.instructor
101
102
103

- $\langle 1, 11, AB, 101 \rangle$

Non-Correlated Query

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O, registration as G
   where O.coursecode = G.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

Inner query
O.instructor
101
102
103

- $\langle 1, 11, AB, 101 \rangle$
  - Since $fid = 101 \in \{101, 102, 103\}$, tuple is returned

Non-Correlated Query

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O, registration as G
   where O.coursecode = G.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

Inner query
O.instructor
101
102
103

- $\langle 1, 11, AB, 101 \rangle$
  - Since $fid = 101 \in \{101, 102, 103\}$, tuple is returned
- $\langle 3, 13, EF, 102 \rangle$

Non-Correlated Query

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O, registration as G
   where O.coursecode = G.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

Inner query
O.instructor
101
102
103

- $\langle 1, 11, AB, 101 \rangle$
  - Since $fid = 101 \in \{101, 102, 103\}$, tuple is returned
- $\langle 3, 13, EF, 102 \rangle$
  - Since $fid = 102 \in \{101, 102, 103\}$, tuple is returned

Non-Correlated Query

```
select S.name
from registration as R, student as S, faculty as F
where S.roll = R.roll and S.name = F.name and F.fid in (
   select O.instructor
   from offering as O, registration as G
   where O.coursecode = G.coursecode );
```

R.roll = S.roll and S.name = F.name			
R.coursecode	roll	name	F.fid
1	11	AB	101
3	13	EF	102

Inner query
O.instructor
101
102
103

- $\langle 1, 11, AB, 101 \rangle$
  - Since $fid = 101 \in \{101, 102, 103\}$, tuple is returned
- $\langle 3, 13, EF, 102 \rangle$
  - Since $fid = 102 \in \{101, 102, 103\}$, tuple is returned
- Thus, non-correlated query results in an error

Set Comparison: some

- $(F\langle \text{comp} \rangle \text{ some } r) \Leftrightarrow (\exists t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < some\{0, 5, 6\} =$

Set Comparison: some

- $(F\langle \text{comp} \rangle \text{ some } r) \Leftrightarrow (\exists t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < some\{0, 5, 6\} = \text{true}$
  - $5 < some\{0, 5\} =$

Set Comparison: some

- $(F\langle \text{comp} \rangle \text{ some } r) \Leftrightarrow (\exists t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < some\{0, 5, 6\} = \text{true}$
  - $5 < some\{0, 5\} = \text{false}$
  - $5 = some\{0, 5\} =$

Set Comparison: some

- $(F\langle \text{comp} \rangle \text{ some } r) \Leftrightarrow (\exists t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < some\{0, 5, 6\} = \text{true}$
  - $5 < some\{0, 5\} = \text{false}$
  - $5 = some\{0, 5\} = \text{true}$
  - $5 \neq some\{0, 5\} =$

Set Comparison: some

- $(F\langle \text{comp} \rangle \text{ some } r) \Leftrightarrow (\exists t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < some\{0, 5, 6\} = \text{true}$
  - $5 < some\{0, 5\} = \text{false}$
  - $5 = some\{0, 5\} = \text{true}$
  - $5 \neq some\{0, 5\} = \text{true}$

Set Comparison: some

- $(F\langle \text{comp} \rangle \text{ some } r) \Leftrightarrow (\exists t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < some\{0, 5, 6\} = \text{true}$
  - $5 < some\{0, 5\} = \text{false}$
  - $5 = some\{0, 5\} = \text{true}$
  - $5 \neq some\{0, 5\} = \text{true}$
- $(= \text{some}) \equiv (\text{in})$
- $(\neq \text{some}) \not\equiv (\text{not in})$

Set Comparison: some

- $(F\langle \text{comp} \rangle \text{ some } r) \Leftrightarrow (\exists t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < some\{0, 5, 6\} = \text{true}$
  - $5 < some\{0, 5\} = \text{false}$
  - $5 = some\{0, 5\} = \text{true}$
  - $5 \neq some\{0, 5\} = \text{true}$
- $(= \text{some}) \equiv (\text{in})$
- $(\neq \text{some}) \not\equiv (\text{not in})$
- Example: Find roll numbers of students who have CPI greater than some student in “CSE”

Set Comparison: some

- $(F\langle \text{comp} \rangle \text{ some } r) \Leftrightarrow (\exists t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < some\{0, 5, 6\} = \text{true}$
  - $5 < some\{0, 5\} = \text{false}$
  - $5 = some\{0, 5\} = \text{true}$
  - $5 \neq some\{0, 5\} = \text{true}$
- $(= \text{some}) \equiv (\text{in})$
- $(\neq \text{some}) \not\equiv (\text{not in})$
- Example: Find roll numbers of students who have CPI greater than some student in "CSE"

```
select roll
from student
where cpi > some (
  select cpi
  from student
  where dept = ''CSE'' );
```

Set Comparison: all

- $(F\langle \text{comp} \rangle \text{ all } r) \Leftrightarrow (\forall t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < all\{0, 5, 6\} =$

Set Comparison: all

- $(F\langle \text{comp} \rangle \text{ all } r) \Leftrightarrow (\forall t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < all\{0, 5, 6\} = \text{false}$
  - $5 < all\{6, 9\} =$

Set Comparison: all

- $(F\langle \text{comp} \rangle \text{ all } r) \Leftrightarrow (\forall t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < all\{0, 5, 6\} = \text{false}$
  - $5 < all\{6, 9\} = \text{true}$
  - $5 = all\{0, 5\} =$

Set Comparison: all

- $(F\langle \text{comp} \rangle \text{ all } r) \Leftrightarrow (\forall t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < all\{0, 5, 6\} = \text{false}$
  - $5 < all\{6, 9\} = \text{true}$
  - $5 = all\{0, 5\} = \text{false}$
  - $5 \neq all\{4, 6\} =$

Set Comparison: all

- $(F\langle \text{comp} \rangle \text{ all } r) \Leftrightarrow (\forall t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < all\{0, 5, 6\} = \text{false}$
  - $5 < all\{6, 9\} = \text{true}$
  - $5 = all\{0, 5\} = \text{false}$
  - $5 \neq all\{4, 6\} = \text{true}$

Set Comparison: all

- $(F\langle \text{comp} \rangle \text{ all } r) \Leftrightarrow (\forall t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < all\{0, 5, 6\} = \text{false}$
  - $5 < all\{6, 9\} = \text{true}$
  - $5 = all\{0, 5\} = \text{false}$
  - $5 \neq all\{4, 6\} = \text{true}$
- $(\neq \text{all}) \equiv (\text{not in})$
- $(= \text{all}) \not\equiv (\text{in})$

Set Comparison: all

- $(F\langle \text{comp} \rangle \text{ all } r) \Leftrightarrow (\forall t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < all\{0, 5, 6\} = \text{false}$
  - $5 < all\{6, 9\} = \text{true}$
  - $5 = all\{0, 5\} = \text{false}$
  - $5 \neq all\{4, 6\} = \text{true}$
- $(\neq \text{all}) \equiv (\text{not in})$
- $(= \text{all}) \not\equiv (\text{in})$
- Example: Find roll numbers of students who have CPI greater than all students in “CSE”

Set Comparison: all

- $(F\langle \text{comp} \rangle \text{ all } r) \Leftrightarrow (\forall t \in r\ (F\langle \text{comp} \rangle t))$
- Examples:
  - $5 < all\{0, 5, 6\} = \text{false}$
  - $5 < all\{6, 9\} = \text{true}$
  - $5 = all\{0, 5\} = \text{false}$
  - $5 \neq all\{4, 6\} = \text{true}$
- $(\neq \text{all}) \equiv (\text{not in})$
- $(= \text{all}) \not\equiv (\text{in})$
- Example: Find roll numbers of students who have CPI greater than all students in "CSE"

```
select roll
from student
where cpi > all (
  select cpi
  from student
  where dept = ''CSE'' );
```

Empty Set

- exists tests if the relation is empty
- (exists r) $\Leftrightarrow$ ($r \neq \Phi$)
- (not exists r) $\Leftrightarrow$ ($r = \Phi$)

Empty Set

- exists tests if the relation is empty
- (exists r) $\Leftrightarrow$ ($r \neq \Phi$)
- (not exists r) $\Leftrightarrow$ ($r = \Phi$)
- Example: Find faculty members who have offered courses in 2018

Empty Set

- exists tests if the relation is empty
- (exists r) $\Leftrightarrow$ ($r \neq \Phi$)
- (not exists r) $\Leftrightarrow$ ($r = \Phi$)
- Example: Find faculty members who have offered courses in 2018

```
select fid
from faculty F
where exists (
  select instructor
  from offering O
  where yr = 2018 and O.instructor = F.fid );
```

Empty Set

- exists tests if the relation is empty
- (exists r) $\Leftrightarrow$ ($r \neq \Phi$)
- (not exists r) $\Leftrightarrow$ ($r = \Phi$)
- Example: Find faculty members who have offered courses in 2018

```
select fid
from faculty F
where exists (
  select instructor
  from offering O
  where yr = 2018 and O.instructor = F.fid );
```

- Example: Find faculty members who have not offered courses in 2018

Empty Set

- exists tests if the relation is empty
- (exists r) $\Leftrightarrow$ ($r \neq \Phi$)
- (not exists r) $\Leftrightarrow$ ($r = \Phi$)
- Example: Find faculty members who have offered courses in 2018

```
select fid
from faculty F
where exists (
  select instructor
  from offering O
  where yr = 2018 and O.instructor = F.fid );
```

- Example: Find faculty members who have not offered courses in 2018

```
...
where not exists (
... );
```

Duplication in Sets

- unique tests if the relation contains duplicate tuples
- (unique r) $\Leftrightarrow$ ($\forall t, s \in r$ ($t \neq s$))
- (not unique r) $\Leftrightarrow$ ($\exists t, s \in r$ ($t = s$))

Duplication in Sets

- unique tests if the relation contains duplicate tuples
- (unique r) $\Leftrightarrow$ ($\forall t, s \in r$ ($t \neq s$))
- (not unique r) $\Leftrightarrow$ ($\exists t, s \in r$ ($t = s$))
- Example: Find faculty members who have only offered one course

Duplication in Sets

- unique tests if the relation contains duplicate tuples
- (unique r) $\Leftrightarrow$ ($\forall t, s \in r$ ($t \neq s$))
- (not unique r) $\Leftrightarrow$ ($\exists t, s \in r$ ($t = s$))
- Example: Find faculty members who have only offered one course

```
select fid
from faculty as F
where unique (
   select coursecode
   from offering O
   where O.instructor = F.fid );
```

- Example: Find faculty members who have offered multiple courses

Duplication in Sets

- unique tests if the relation contains duplicate tuples
- (unique r) $\Leftrightarrow$ ($\forall t, s \in r$ ($t \neq s$))
- (not unique r) $\Leftrightarrow$ ($\exists t, s \in r$ ($t = s$))
- Example: Find faculty members who have only offered one course

```
select fid
from faculty as F
where unique (
   select coursecode
   from offering O
   where O.instructor = F.fid );
```

- Example: Find faculty members who have offered multiple courses

```
...
where not unique (
... );
```

Explicit Sets

- Use set literals specified within brackets
- Example: Find students in “CSE” and “ECO” 33

```
select *
from student
where dept in (''CSE'', ''ECO'');
```

Summary of SQL Query Format

- May contain up to six clauses
- May be nested
- Only the first two, select and from, are mandatory
- Format (in order)
  select ⟨ attribute list ⟩
  from ⟨ relation list ⟩
  where ⟨ predicate or tuple condition ⟩
  group by ⟨ group attribute list ⟩
  having ⟨ group condition ⟩
  order by ⟨ attribute list ⟩

Summary of SQL Query Format

- May contain up to six clauses
- May be nested
- Only the first two, select and from, are mandatory
- Format (in order)
  select ⟨ attribute list ⟩
  from ⟨ relation list ⟩
  where ⟨ predicate or tuple condition ⟩
  group by ⟨ group attribute list ⟩
  having ⟨ group condition ⟩
  order by ⟨ attribute list ⟩
- Execution order

Summary of SQL Query Format

- May contain up to six clauses
- May be nested
- Only the first two, select and from, are mandatory
- Format (in order)
  select ⟨ attribute list ⟩
  from ⟨ relation list ⟩
  where ⟨ predicate or tuple condition ⟩
  group by ⟨ group attribute list ⟩
  having ⟨ group condition ⟩
  order by ⟨ attribute list ⟩
- Execution order
  1. from

Summary of SQL Query Format

- May contain up to six clauses
- May be nested
- Only the first two, select and from, are mandatory
- Format (in order)
  select ⟨ attribute list ⟩
  from ⟨ relation list ⟩
  where ⟨ predicate or tuple condition ⟩
  group by ⟨ group attribute list ⟩
  having ⟨ group condition ⟩
  order by ⟨ attribute list ⟩
- Execution order
  1. from
  2. where

Summary of SQL Query Format

- May contain up to six clauses
- May be nested
- Only the first two, select and from, are mandatory
- Format (in order)
  select ⟨ attribute list ⟩
  from ⟨ relation list ⟩
  where ⟨ predicate or tuple condition ⟩
  group by ⟨ group attribute list ⟩
  having ⟨ group condition ⟩
  order by ⟨ attribute list ⟩
- Execution order
  1. from
  2. where
  3. group by

Summary of SQL Query Format

- May contain up to six clauses
- May be nested
- Only the first two, select and from, are mandatory
- Format (in order)
  select ⟨ attribute list ⟩
  from ⟨ relation list ⟩
  where ⟨ predicate or tuple condition ⟩
  group by ⟨ group attribute list ⟩
  having ⟨ group condition ⟩
  order by ⟨ attribute list ⟩
- Execution order
  1. from
  2. where
  3. group by
  4. having

Summary of SQL Query Format

- May contain up to six clauses
- May be nested
- Only the first two, select and from, are mandatory
- Format (in order)
  select ⟨ attribute list ⟩
  from ⟨ relation list ⟩
  where ⟨ predicate or tuple condition ⟩
  group by ⟨ group attribute list ⟩
  having ⟨ group condition ⟩
  order by ⟨ attribute list ⟩
- Execution order
  1. from
  2. where
  3. group by
  4. having
  5. select

Summary of SQL Query Format

- May contain up to six clauses
- May be nested
- Only the first two, select and from, are mandatory
- Format (in order)
  select ⟨ attribute list ⟩
  from ⟨ relation list ⟩
  where ⟨ predicate or tuple condition ⟩
  group by ⟨ group attribute list ⟩
  having ⟨ group condition ⟩
  order by ⟨ attribute list ⟩
- Execution order
  1. from
  2. where
  3. group by
  4. having
  5. select
  6. order by

Derived Relations

- In the from clause, a derived relation (result of a subquery) can be used

Derived Relations

- In the from clause, a derived relation (result of a subquery) can be used
- Example: Find departments and average CPIs where average CPI is greater than 8.0

Derived Relations

- In the from clause, a derived relation (result of a subquery) can be used
- Example: Find departments and average CPIs where average CPI is greater than 8.0

```
select deptid, avg_cpi
from (
   select dept, avg(cpi)
   from student
   group by dept )
     as dept_avg (deptid, avg_cpi)
where avg_cpi >= 8.0;
```

Derived Relations

- In the from clause, a derived relation (result of a subquery) can be used
- Example: Find departments and average CPIs where average CPI is greater than 8.0

```
select deptid, avg_cpi
from (
   select dept, avg(cpi)
   from student
   group by dept )
     as dept_avg (deptid, avg_cpi)
where avg_cpi >= 8.0;
```

- Avoids using having clause

With

- with clause defines a temporary relation
- This temporary relation is available *only* to the query using the with clause

With

- with clause defines a temporary relation
- This temporary relation is available *only* to the query using the with clause
- Example: Find departments and average CPIs where average CPI is greater than 8.0

With

- with clause defines a temporary relation
- This temporary relation is available *only* to the query using the with clause
- Example: Find departments and average CPIs where average CPI is greater than 8.0

```
with dept_avg (deptid, avg_cpi) as
   select dept, avg(cpi)
   from student
   group by dept
select deptid, avg_cpi
from dept_avg
where avg_cpi >= 8.0;
```

Insertion

Insertion

- insert into . . . values statement

Insertion

- insert into . . . values statement
- Example: Create a new student “ABC” with roll 1897 and department 7

Insertion

- insert into . . . values statement
- Example: Create a new student “ABC” with roll 1897 and department 7

```
insert into student(roll, name, dept, cpi)
values (1897, ''ABC'', 7, 0.0);
```

Insertion

- insert into . . . values statement
- Example: Create a new student “ABC” with roll 1897 and department 7

```
insert into student(roll, name, dept, cpi)
values (1897, ‘‘ABC’’, 7, 0.0);
```

- May omit schema

```
insert into student
values (1897, ‘‘ABC’’, 7, 0.0);
```

Insertion

- insert into . . . values statement
- Example: Create a new student “ABC” with roll 1897 and department 7

```
insert into student(roll, name, dept, cpi)
values (1897, ‘‘ABC’’, 7, 0.0);
```

- May omit schema

```
insert into student
values (1897, ‘‘ABC’’, 7, 0.0);
```

- If value is not known, specify *null*

```
insert into student
values (1897, ‘‘ABC’’, 7, null);
```

Insertion

- insert into . . . values statement
- Example: Create a new student “ABC” with roll 1897 and department 7

```
insert into student(roll, name, dept, cpi)
values (1897, ''ABC'', 7, 0.0);
```

- May omit schema

```
insert into student
values (1897, ''ABC'', 7, 0.0);
```

- If value is not known, specify *null*

```
insert into student
values (1897, ''ABC'', 7, null);
```

- To avoid *null*, specify schema

```
insert into student(roll, name, dept)
values (1897, ''ABC'', 7);
```

Insertion (contd.)

- Example: Create a course of code 9 for every department with the same type

Insertion (contd.)

- Example: Create a course of code 9 for every department with the same type

```
insert into course(code, title, webpage, ctype)
select 9, ''New'', null, type
from course
where type in (
  select deptid
  from department );
```

- Query is evaluated fully before any tuple is inserted

Insertion (contd.)

- Example: Create a course of code 9 for every department with the same type

```
insert into course(code, title, webpage, ctype)
select 9, ''New'', null, type
from course
where type in (
  select deptid
  from department );
```

- Query is evaluated fully before any tuple is inserted
- Otherwise, infinite insertion happens for queries like

```
insert into r
select * from r;
```

Deletion

- delete from . . . where statement

Deletion

- delete from . . . where statement
- Example: Delete student with roll number 1946

Deletion

- delete from . . . where statement
- Example: Delete student with roll number 1946

```
delete from student
where roll = 1946;
```

- where selects tuples that will be deleted

Deletion

- delete from . . . where statement
- Example: Delete student with roll number 1946

```
delete from student
where roll = 1946;
```

- where selects tuples that will be deleted
- If where is empty,

Deletion

- delete from . . . where statement
- Example: Delete student with roll number 1946

```
delete from student
where roll = 1946;
```

- where selects tuples that will be deleted
- If where is empty, all tuples are deleted

Deletion

- delete from . . . where statement
- Example: Delete student with roll number 1946

```
delete from student
where roll = 1946;
```

- where selects tuples that will be deleted
- If where is empty, all tuples are deleted
- Delete all students

```
delete from student;
```

Deletion (contd.)

- Example: Delete all students whose CPI is less than the average CPI

Deletion (contd.)

- Example: Delete all students whose CPI is less than the average CPI

```
delete from student
where cpi < (
   select avg(cpi)
   from student );
```

Deletion (contd.)

- Example: Delete all students whose CPI is less than the average CPI

```
delete from student
where cpi < (
   select avg(cpi)
   from student );
```

- Average is computed before any tuple is deleted
- It is *not* re-computed

Deletion (contd.)

- Example: Delete all students whose CPI is less than the average CPI

```
delete from student
where cpi < (
   select avg(cpi)
   from student );
```

- Average is computed before any tuple is deleted
- It is *not* re-computed
- Otherwise, average keeps changing
- Ultimately, only the student with the largest CPI remains

Updating

- update . . . set . . . where statement
- where selects tuples that will be updated

Updating

- update . . . set . . . where statement
- where selects tuples that will be updated
- Example: Update value of grade ‘E’ to 2

Updating

- update . . . set . . . where statement
- where selects tuples that will be updated
- Example: Update value of grade ‘E’ to 2

```
update grade
set value = 2.0
where gradecode = ‘E’;
```

Updating

- update . . . set . . . where statement
- where selects tuples that will be updated
- Example: Update value of grade ‘E’ to 2

```
update grade
set value = 2.0
where gradecode = ‘E’;
```

- If where is empty, all tuples are updated with the new value
- Example: Increase CPI of all students by 5%

Updating

- update . . . set . . . where statement
- where selects tuples that will be updated
- Example: Update value of grade ‘E’ to 2

```
update grade
set value = 2.0
where gradecode = ‘E’;
```

- If where is empty, all tuples are updated with the new value
- Example: Increase CPI of all students by 5%

```
update student
set cpi = cpi * 1.05;
```

Updating (contd.)

- Example: Increase CPI of all students by 10% where CPI is less than 6.0 and by 5% otherwise

```
update student set cpi = cpi * 1.05 where cpi >= 6.0;
update student set cpi = cpi * 1.10 where cpi < 6.0;
```

Updating (contd.)

- Example: Increase CPI of all students by 10% where CPI is less than 6.0 and by 5% otherwise

```
update student set cpi = cpi * 1.05 where cpi >= 6.0;
update student set cpi = cpi * 1.10 where cpi < 6.0;
```

- Order of statements is important

Updating (contd.)

- Example: Increase CPI of all students by 10% where CPI is less than 6.0 and by 5% otherwise

```
update student set cpi = cpi * 1.05 where cpi >= 6.0;
update student set cpi = cpi * 1.10 where cpi < 6.0;
```

- Order of statements is important
- case statement handles conditional updates in a better manner and is sometimes necessary

Updating (contd.)

- Example: Increase CPI of all students by 10% where CPI is less than 6.0 and by 5% otherwise

```
update student set cpi = cpi * 1.05 where cpi >= 6.0;
update student set cpi = cpi * 1.10 where cpi < 6.0;
```

- Order of statements is important
- case statement handles conditional updates in a better manner and is sometimes necessary
- Example: Increase CPI of all students by 10% where CPI is less than 6.0, by 5% when less than 8.0, and 2% otherwise

Updating (contd.)

- Example: Increase CPI of all students by 10% where CPI is less than 6.0 and by 5% otherwise

```
update student set cpi = cpi * 1.05 where cpi >= 6.0;
update student set cpi = cpi * 1.10 where cpi < 6.0;
```

- Order of statements is important
- case statement handles conditional updates in a better manner and is sometimes necessary
- Example: Increase CPI of all students by 10% where CPI is less than 6.0, by 5% when less than 8.0, and 2% otherwise

```
update student
set cpi =
  case (cpi)
    when cpi < 6.0 then cpi * 1.10
    when cpi < 8.0 then cpi * 1.05
    else cpi * 1.02
  end;
```

Join

- Join types: inner join, left (outer) join, right (outer) join, full (outer) join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

```
student inner join program on student.roll = program.roll;
```

Join

- Join types: inner join, left (outer) join, right (outer) join, full (outer) join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

```
student inner join program on student.roll = program.roll;
student natural left join program;
```

Join

- Join types: inner join, left (outer) join, right (outer) join, full (outer) join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

```
student inner join program on student.roll = program.roll;
student natural left join program;
student right outer join program using (roll);
```

Join

- Join types: inner join, left (outer) join, right (outer) join, full (outer) join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

```
student inner join program on student.roll = program.roll;
student natural left join program;
student right outer join program using (roll);
```

- Inner natural join is assumed when nothing is mentioned

Join

- Join types: inner join, left (outer) join, right (outer) join, full (outer) join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

```
student inner join program on student.roll = program.roll;
student natural left join program;
student right outer join program using (roll);
```

- Inner natural join is assumed when nothing is mentioned
- Multiple relations can be joined

```
student join program join registration;
```

Views

- A relation that is not present physically but is made visible to the user is called a view
- A view is a *virtual* relation derived from other relations

Views

- A relation that is not present physically but is made visible to the user is called a view
- A view is a *virtual* relation derived from other relations
- It helps in query processing
  - If a sub-query is very common, obtain a view for it
- It helps in hiding certain data from a user
  - A view can leave out sensitive attributes
  - Example:

```
create view student_program as
  student natural join program;
```

Views

- A relation that is not present physically but is made visible to the user is called a view
- A view is a *virtual* relation derived from other relations
- It helps in query processing
  - If a sub-query is very common, obtain a view for it
- It helps in hiding certain data from a user
  - A view can leave out sensitive attributes
  - Example:

```
create view student_program as
  student natural join program;
```

- A view can be deleted simply using drop

```
drop student_program;
```

- A view has full query capabilities, but limited modification facilities
- A view can be defined using other views, but not itself

Storing Views

- A view is *not* stored physically
- Only the query expression is stored
- Wherever a view is used, the query expression is substituted

Storing Views

- A view is *not* stored physically
- Only the query expression is stored
- Wherever a view is used, the query expression is substituted
- Example: Find students and corresponding programs for “CSE”

Storing Views

- A view is *not* stored physically
- Only the query expression is stored
- Wherever a view is used, the query expression is substituted
- Example: Find students and corresponding programs for "CSE"

```
select *
from student_program
where dept = ''CSE'';
```

is expanded at runtime to

Storing Views

- A view is *not* stored physically
- Only the query expression is stored
- Wherever a view is used, the query expression is substituted
- Example: Find students and corresponding programs for "CSE"

```
select *
from student_program
where dept = ''CSE'';
```

is expanded at runtime to

```
select *
from
  ( student natural join program )
where dept = ''CSE'';
```

Storing Views

- A view is *not* stored physically
- Only the query expression is stored
- Wherever a view is used, the query expression is substituted
- Example: Find students and corresponding programs for "CSE"

```
select *
from student_program
where dept = ''CSE'';
```

is expanded at runtime to

```
select *
from
  ( student natural join program )
where dept = ''CSE'';
```

- This allows to capture all updates in the base relations

Storing Views

- A view is *not* stored physically
- Only the query expression is stored
- Wherever a view is used, the query expression is substituted
- Example: Find students and corresponding programs for "CSE"

```
select *
from student_program
where dept = ''CSE'';
```

is expanded at runtime to

```
select *
from
  ( student natural join program )
where dept = ''CSE'';
```

- This allows to capture all updates in the base relations
- If a view is materialized, it is stored physically
- To ensure consistency, database *must* update materialized views once base relations are updated

Updating a View

- Updating a view causes many problems, and is, in general, not allowed

Updating a View

- Updating a view causes many problems, and is, in general, not allowed
- Update must map to updates on the base relations

Updating a View

- Updating a view causes many problems, and is, in general, not allowed
- Update must map to updates on the base relations
- If a view involves join or Cartesian product, update must map to updates on all the base relations

Updating a View

- Updating a view causes many problems, and is, in general, not allowed
- Update must map to updates on the base relations
- If a view involves join or Cartesian product, update must map to updates on all the base relations
  - Not always possible

Updating a View

- Updating a view causes many problems, and is, in general, not allowed
- Update must map to updates on the base relations
- If a view involves join or Cartesian product, update must map to updates on all the base relations
  - Not always possible
- Problems with insert or delete

Updating a View

- Updating a view causes many problems, and is, in general, not allowed
- Update must map to updates on the base relations
- If a view involves join or Cartesian product, update must map to updates on all the base relations
  - Not always possible
- Problems with insert or delete
  - Spurious tuple
  - Null
  - Non-uniqueness

Triggers

- A trigger statement allows *automatic* and *active* management during database modifications
- It is invoked *only* by the database engine and not by the user

Triggers

- A trigger statement allows *automatic* and *active* management during database modifications
- It is invoked *only* by the database engine and not by the user
- It follows the event-condition-action (ECA) model
  - Event: Database modification
  - Condition: Invoked only if true; if no condition, then assumed true
  - Action: Database action or any program

Triggers

- A trigger statement allows *automatic* and *active* management during database modifications
- It is invoked *only* by the database engine and not by the user
- It follows the event-condition-action (ECA) model
  - Event: Database modification
  - Condition: Invoked only if true; if no condition, then assumed true
  - Action: Database action or any program
- It may not allow the full range of modification statements

Triggers

- A trigger statement allows *automatic* and *active* management during database modifications
- It is invoked *only* by the database engine and not by the user
- It follows the event-condition-action (ECA) model
  - Event: Database modification
  - Condition: Invoked only if true; if no condition, then assumed true
  - Action: Database action or any program
- It may not allow the full range of modification statements
- It can be called *before* or *after* the modification
- New and old values are referenced using new and old keywords
  - new refers to a inserted or new value of updated tuple
  - old refers to a deleted or old value of updated tuple
- By default, it is for each row (i.e., tuple)

Creating Triggers

- Created using a create trigger command

Creating Triggers

- Created using a create trigger command
- Example: Update the coursecode of offering when the code of a course of type 9 is updated

Creating Triggers

- Created using a create trigger command
- Example: Update the coursecode of offering when the code of a course of type 9 is updated

```
create trigger update_code
  after update of code on course
  for each row
  when ctype = 9
  begin
    update offering set coursecode = new.code where
        coursecode = old.code;
  end;
```

Creating Triggers

- Created using a create trigger command
- Example: Update the coursecode of offering when the code of a course of type 9 is updated

```
create trigger update_code
  after update of code on course
  for each row
  when ctype = 9
  begin
    update offering set coursecode = new.code where
        coursecode = old.code;
  end;
```

- A trigger can be deleted simply using drop

```
drop update_code;
```

Indices in SQL

- create [unique] index name on r (a, b) creates an index *name* on attributes *a, b* of relation *r*
  - *unique* does not allow duplicates on the atrributes

```
create index idx_ctype on course (ctype);
```

Indices in SQL

- create [unique] index name on r (a, b) creates an index *name* on attributes *a, b* of relation *r*
  - *unique* does not allow duplicates on the atrributes

```
create index idx_ctype on course (ctype);
```

- Index will be used whenever attribute *a* of relation *r* is used
- Often, primary keys are implicity indexed

Indices in SQL

- create [unique] index name on r (a, b) creates an index *name* on attributes *a, b* of relation *r*
  - *unique* does not allow duplicates on the atrributes

```
create index idx_ctype on course (ctype);
```

- Index will be used whenever attribute *a* of relation *r* is used
- Often, primary keys are implicity indexed
- Slows down modification operations as index is also modified

Indices in SQL

- create [unique] index name on r (a, b) creates an index *name* on attributes *a, b* of relation *r*
  - *unique* does not allow duplicates on the atrributes

```
create index idx_ctype on course (ctype);
```

- Index will be used whenever attribute *a* of relation *r* is used
- Often, primary keys are implicity indexed
- Slows down modification operations as index is also modified
- drop index *i* deletes the index

```
drop index idx_ctype;
```

Access Control

- SQL is a data control language (DCL)
- Can be used to control accesses of users to data
- *Data security*

Access Control

- SQL is a data control language (DCL)
- Can be used to control accesses of users to data
- *Data security*
- grant: grant privilege on object to user [with grant option]
  - *privilege*: Can be all or specific:
    - System privileges on tables and views: create, alter, drop
    - Object privileges: select, insert/update/delete, execute
  - *object*: table, view, stored procedure
  - *user*: public or particular username or *role*
  - *with grant option*: can grant access rights to others

Access Control

- SQL is a data control language (DCL)
- Can be used to control accesses of users to data
- *Data security*
- grant: grant privilege on object to user [with grant option]
  - *privilege*: Can be all or specific:
    - System privileges on tables and views: create, alter, drop
    - Object privileges: select, insert/update/delete, execute
  - *object*: table, view, stored procedure
  - *user*: public or particular username or *role*
  - *with grant option*: can grant access rights to others
- revoke: revoke privilege on object from user

Access Control

- SQL is a data control language (DCL)
- Can be used to control accesses of users to data
- *Data security*
- grant: grant privilege on object to user [with grant option]
  - *privilege*: Can be all or specific:
    - System privileges on tables and views: create, alter, drop
    - Object privileges: select, insert/update/delete, execute
  - *object*: table, view, stored procedure
  - *user*: public or particular username or *role*
  - *with grant option*: can grant access rights to others
- revoke: revoke privilege on object from user

```
grant select on student to xyz;
grant all on course to abc with grant option;
revoke create on student from xyz;
```

Transaction Control

- Transactions are groups of statements that are executed atomically

Transaction Control

- Transactions are groups of statements that are executed atomically
- set transaction [read write | read only] starts a transaction

Transaction Control

- Transactions are groups of statements that are executed atomically
- set transaction [read write | read only] starts a transaction
- After modification operations, a transaction can commit or rollback

Transaction Control

- Transactions are groups of statements that are executed atomically
- set transaction [read write | read only] starts a transaction
- After modification operations, a transaction can commit or rollback
- Within a transaction, a checkpoint can be set by savepoint savepoint-name

Transaction Control

- Transactions are groups of statements that are executed atomically
- set transaction [read write | read only] starts a transaction
- After modification operations, a transaction can commit or rollback
- Within a transaction, a checkpoint can be set by savepoint savepoint-name
- A transaction can rollback to a particular savepoint-name

Transaction Control

- Transactions are groups of statements that are executed atomically
- set transaction [read write | read only] starts a transaction
- After modification operations, a transaction can commit or rollback
- Within a transaction, a checkpoint can be set by savepoint savepoint-name
- A transaction can rollback to a particular savepoint-name
- A savepoint can be removed by release savepoint savepoint-name

Transaction Control

- Transactions are groups of statements that are executed atomically
- set transaction [read write | read only] starts a transaction
- After modification operations, a transaction can commit or rollback
- Within a transaction, a checkpoint can be set by savepoint savepoint-name
- A transaction can rollback to a particular savepoint-name
- A savepoint can be removed by release savepoint savepoint-name

roll	name
1	AB
2	CD

```
set transaction read write;
delete from student where roll = 1;
commit;
```

Transaction Control

- Transactions are groups of statements that are executed atomically
- set transaction [read write | read only] starts a transaction
- After modification operations, a transaction can commit or rollback
- Within a transaction, a checkpoint can be set by savepoint savepoint-name
- A transaction can rollback to a particular savepoint-name
- A savepoint can be removed by release savepoint savepoint-name

roll	name
1	AB
2	CD

```
set transaction read write;
delete from student where roll = 1;
commit;
```

roll	name
2	CD

Savepoint Example

roll	name
1	AB
2	CD
3	EF
4	GH

```
set transaction read write;
savepoint sp1;
delete from student where roll = 1;
savepoint sp2;
delete from student where roll = 2;
rollback to sp2;
```

Savepoint Example

roll	name
1	AB
2	CD
3	EF
4	GH

```
set transaction read write;
savepoint sp1;
delete from student where roll = 1;
savepoint sp2;
delete from student where roll = 2;
rollback to sp2;
```

roll	name
2	CD
3	EF
4	GH

Variants in SQL

- SQL standards have evolved a lot over the years
- Different vendors provide different flavors and may not implement every feature