CS251/CS253: Computing Tools
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/

5th February, 2020

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

Relational Algebra

- SQL is based upon relational algebra
- Relational data model
- Operators are functions from one or two input relations to an output relation

Relational Algebra

- SQL is based upon relational algebra
- Relational data model
- Operators are functions from one or two input relations to an output relation
- Uses *propositional calculus* formed by *expressions* connected by
  1. and: $\wedge$
  2. or: $\vee$
  3. not: $\neg$
- Each term is of the form
  `<attr/const> comparator <attr/const>`
  where `comparator` is one of $=, \neq, >, \geq, <, \leq$

Relation

- A relation is a subset of the cross-product of sets
- For sets $D_1, D_2, \ldots, D_n$, a relation r is a set of n-ary tuples of the form $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- Example
  - `name` = {A, B, C}
  - `designation` = {L, E, W}
  - `identifier` = $\mathcal{N}$
  - r = {(A, E, 4), (B, E, 9), (C, W, 23)} is a relation over `name` × `designation` × `identifier`
- Relations are *unordered*

Relation

- A relation is a subset of the cross-product of sets
- For sets $D_1, D_2, \ldots, D_n$, a relation r is a set of n-ary tuples of the form $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- Example
  - `name` = {A, B, C}
  - `designation` = {L, E, W}
  - `identifier` = $\mathcal{N}$
  - r = {(A, E, 4), (B, E, 9), (C, W, 23)} is a relation over `name` × `designation` × `identifier`
- Relations are *unordered*
- Generally depicted as a table

name	designation	identifier
A	E	4
B	E	9
C	W	23

Attribute

- Each attribute of a relation has a name
- There is a domain for each attribute
- Attributes are *generally* atomic
  - Indivisible, not sets
- Domain is atomic if all members are atomic
- Special value null in every domain

Relation Schema and Tuple

- The sets define a relation schema
- Example
  - Schema is `Person = (name, designation, identifier)`
- Relations are defined over a schema
- If schema is R, relation is denoted by $r(R)$
  - Example: `persons(Person)`
- A relation instance is a particular instance from the schema
  - Earlier example
- An element of a relation (instance) is a tuple

Relation Schema and Tuple

- The sets define a relation schema
- Example
  - Schema is `Person = (name, designation, identifier)`
- Relations are defined over a schema
- If schema is R, relation is denoted by $r(R)$
  - Example: `persons(Person)`
- A relation instance is a particular instance from the schema
  - Earlier example
- An element of a relation (instance) is a tuple
- Tuples are rows and attributes are columns

Database

- Consists of multiple *inter-related* relations
- Each relation stores information about a particular relationship

Database

- Consists of multiple *inter-related* relations
- Each relation stores information about a particular relationship
- Alternatively, a single relation can store all data
- Problems
  - Data repetition
  - Need for null values
- Normalization theory deals with how to design relation schemas

Key

- $K \subseteq R$ is a superkey of R if and only if values for K are sufficient to identify a unique tuple in *all* possible relations $r(R)$
  - Possible $r(R)$ signifies a relation that can exist from the data that is being modeled
- Example: {name} is a superkey if each person has a unique name, otherwise not
- All supersets of superkeys are superkeys
  - {name, designation} is also a superkey
- In practical situations, an id will be used which is guaranteed to be a superkey

Key

- $K \subseteq R$ is a superkey of R if and only if values for K are sufficient to identify a unique tuple in *all* possible relations $r(R)$
  - Possible $r(R)$ signifies a relation that can exist from the data that is being modeled
- Example: {name} is a superkey if each person has a unique name, otherwise not
- All supersets of superkeys are superkeys
  - {name, designation} is also a superkey
- In practical situations, an id will be used which is guaranteed to be a superkey
- A superkey K is a candidate key if K is minimal, i.e., no proper subset of it is a superkey
  - {name} is a candidate key

Key

- $K \subseteq R$ is a superkey of R if and only if values for K are sufficient to identify a unique tuple in *all* possible relations $r(R)$
  - Possible $r(R)$ signifies a relation that can exist from the data that is being modeled
- Example: {name} is a superkey if each person has a unique name, otherwise not
- All supersets of superkeys are superkeys
  - {name, designation} is also a superkey
- In practical situations, an id will be used which is guaranteed to be a superkey
- A superkey K is a candidate key if K is minimal, i.e., no proper subset of it is a superkey
  - {name} is a candidate key
- There may be multiple candidate keys
  - {name}, {identifier} are candidate keys

Key

- $K \subseteq R$ is a superkey of R if and only if values for K are sufficient to identify a unique tuple in *all* possible relations $r(R)$
  - Possible $r(R)$ signifies a relation that can exist from the data that is being modeled
- Example: {name} is a superkey if each person has a unique name, otherwise not
- All supersets of superkeys are superkeys
  - {name, designation} is also a superkey
- In practical situations, an id will be used which is guaranteed to be a superkey
- A superkey K is a candidate key if K is minimal, i.e., no proper subset of it is a superkey
  - {name} is a candidate key
- There may be multiple candidate keys
  - {name}, {identifier} are candidate keys
- Primary key is a candidate key chosen to serve as the primary means of identifying tuples
  - Choice is arbitrary as it depends on the database designer
  - Other candidate keys are called secondary keys

Foreign Key

- A relation schema may have an attribute that is unique (e.g., a primary key) in another schema
- This attribute is then called a foreign key
- Example
  - depositor = (name, number)
  - customer = (name, street, city)
  - account = (number, balance)
  - name and number in depositor are foreign keys
- Values in the foreign key attribute of the referencing relation may only come from those in the primary key of the referenced relation

Example Schema

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

- 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

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),
  primary key fid,
  foreign key (dept) references department,
  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 $A_1, \dots, A_n$
  from $r_1, \dots, r_m$
  where P
- Each r_i is a relation
- Each A_j is an attribute from one of $r_1, \dots, r_m$
- P is a predicate involving attributes and constants
- where can be left out, which then means *true*
- Result is a relation with the schema $(A_1, \dots, A_n)$
- Is equivalent to the relational algebra query
  $\Pi_{A_1,\dots,A_n}(\sigma_P(r_1 \times \cdots \times r_m))$

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*

Project in Relational Algebra

- $\Pi_{A_1,\ldots,A_k}(r)$
- A_i, etc. are attributes of r
- Select only the specified attributes $A_1, \ldots, A_k$ from all tuples of r
- Duplicate rows are removed, since relations are sets

Project in Relational Algebra

- $\Pi_{A_1,\ldots,A_k}(r)$
- A_i, etc. are attributes of r
- Select only the specified attributes $A_1, \ldots, A_k$ from all tuples of r
- Duplicate rows are removed, since relations are sets
- Schema is

Project in Relational Algebra

- $\Pi_{A_1,\ldots,A_k}(r)$
- A_i, etc. are attributes of r
- Select only the specified attributes $A_1, \ldots, A_k$ from all tuples of r
- Duplicate rows are removed, since relations are sets
- Schema is changed
- Applying $\Pi_{A,C}$ on

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

Project in Relational Algebra

- $\Pi_{A_1,\ldots,A_k}(r)$
- A_i, etc. are attributes of r
- Select only the specified attributes $A_1, \ldots, A_k$ from all tuples of r
- Duplicate rows are removed, since relations are sets
- Schema is changed
- Applying $\Pi_{A,C}$ on

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

A	C
1	5
2	5
2	8

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;
```

Cartesian Product in Relational Algebra

- $r \times s = \{\langle u, v\rangle | u \in r \text{ and } v \in s\}$
- Attributes of relations *r* and *s* should be disjoint
- If attributes are not disjoint, renaming should be used
- Schema is

Cartesian Product in Relational Algebra

- $r \times s = \{\langle u, v \rangle | u \in r \text{ and } v \in s\}$
- Attributes of relations r and s should be disjoint
- If attributes are not disjoint, renaming should be used
- Schema is changed
- Applying $\times$ on

A	B
1	1
2	2

and

C	D	E
1	2	7
2	6	8
5	7	9

returns

Cartesian Product in Relational Algebra

- $r \times s = \{\langle u, v\rangle | u \in r \text{ and } v \in s\}$
- Attributes of relations *r* and *s* should be disjoint
- If attributes are not disjoint, renaming should be used
- Schema is changed
- Applying $\times$ on

A	B
1	1
2	2

and

C	D	E
1	2	7
2	6	8
5	7	9

returns

A	B	C	D	E
1	1	1	2	7
1	1	2	6	8
1	1	5	7	9
2	2	1	2	7
2	2	2	6	8
2	2	5	7	9

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

```sql
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 roll number 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 roll number of students in B.Tech. program

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

Select in Relational Algebra

- $\sigma_p(r) = \{t | t \in r \text{ and } p(t)\}$
- p is called the selection predicate
- Select all tuples from r that satisfies the predicate p
- Schema is

Select in Relational Algebra

- $\sigma_p(r) = \{t | t \in r \text{ and } p(t)\}$
- p is called the selection predicate
- Select all tuples from r that satisfies the predicate p
- Schema is not changed
- Applying $\sigma_{A=B \wedge D>5}$ on

A	B	C	D
1	1	2	7
1	2	5	7
2	2	9	3
2	2	8	6

returns

Select in Relational Algebra

- $\sigma_p(r) = \{t | t \in r \text{ and } p(t)\}$
- p is called the selection predicate
- Select all tuples from r that satisfies the predicate p
- Schema is not changed
- Applying $\sigma_{A=B \wedge D>5}$ on

A	B	C	D
1	1	2	7
1	2	5	7
2	2	9	3
2	2	8	6

returns

A	B	C	D
1	1	2	7
2	2	8	6

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

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

- May use and, or and not to connect predicates

```sql
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'';
```

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

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

- Use desc to obtain tuples in descending order

```sql
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);
```

Aggregation in Relational Algebra

- Aggregate functions that can be used are *avg*, *min*, *max*, *sum*, *count*
- Can be applied on groups of tuples as well
- Aggregate operation is of the form ${}_{G_1,\dots,G_k}\mathcal{G}_{F_1(A_1),\dots,F_n(A_n)}(E)$ where
  - $G_1, \dots, G_k$ is the list of attributes on which to group (may be empty)
  - Each F_i is an aggregate function that operates on the attribute A_i
- Applying $\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

Aggregation in Relational Algebra

- Aggregate functions that can be used are *avg*, *min*, *max*, *sum*, *count*
- Can be applied on groups of tuples as well
- Aggregate operation is of the form ${}_{G_1,\ldots,G_k}\mathcal{G}_{F_1(A_1),\ldots,F_n(A_n)}(E)$ where
  - $G_1, \ldots, G_k$ is the list of attributes on which to group (may be empty)
  - Each F_i is an aggregate function that operates on the attribute A_i
- Applying $\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8

returns

sum(C)
23

Grouping and Aggregation in Relational Algebra

- First, the tuples are grouped according to $G_1, \ldots, G_k$
- Then, aggregate functions $F_1(A_1), \ldots, F_n(A_n)$ are applied on each group
- Schema changes to $(G_1, \ldots, G_k, F_1(A_1), \ldots, F_n(A_n))$
- Applying ${}_A\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8
3	4	8

returns

Grouping and Aggregation in Relational Algebra

- First, the tuples are grouped according to $G_1, \ldots, G_k$
- Then, aggregate functions $F_1(A_1), \ldots, F_n(A_n)$ are applied on each group
- Schema changes to $(G_1, \ldots, G_k, F_1(A_1), \ldots, F_n(A_n))$
- Applying ${}_A\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8
3	4	8

returns

A	sum(C)
1	10
2	13
3	8

Grouping and Aggregation in Relational Algebra

- First, the tuples are grouped according to $G_1, \ldots, G_k$
- Then, aggregate functions $F_1(A_1), \ldots, F_n(A_n)$ are applied on each group
- Schema changes to $(G_1, \ldots, G_k, F_1(A_1), \ldots, F_n(A_n))$
- Applying ${}_{A}\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8
3	4	8

returns

A	sum(C)
1	10
2	13
3	8

- Applying ${}_{A,B}\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
1	2	4

returns

Grouping and Aggregation in Relational Algebra

- First, the tuples are grouped according to $G_1, \ldots, G_k$
- Then, aggregate functions $F_1(A_1), \ldots, F_n(A_n)$ are applied on each group
- Schema changes to $(G_1, \ldots, G_k, F_1(A_1), \ldots, F_n(A_n))$
- Applying ${}_A\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
2	3	5
2	4	8
3	4	8

returns

A	sum(C)
1	10
2	13
3	8

- Applying ${}_{A,B}\mathcal{G}_{sum(C)}$ on r

A	B	C
1	1	5
1	2	5
1	2	4

returns

A	B	sum(C)
1	1	5
1	2	9

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

```sql
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 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

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

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
- Inner query is evaluated for *each tuple* in the outer query
- When a nested query refers to an attribute in the outer query, it is called a correlated query

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 );
```

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 ⟩

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

```
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 in Relational Algebra

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is

Join in Relational Algebra

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is same as $r \times s$ but (potentially) less number of tuples
- The above form is the most general, called the theta join

Join in Relational Algebra

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is same as $r \times s$ but (potentially) less number of tuples
- The above form is the most general, called the theta join
- Equality join: When the join condition only contains equality
  - $r \bowtie_{B=C} s$

Join in Relational Algebra

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is same as $r \times s$ but (potentially) less number of tuples
- The above form is the most general, called the theta join
- Equality join: When the join condition only contains equality
  - $r \bowtie_{B=C} s$
- Natural join: If two relations share an attribute (also its *name*), equality join on that common attribute
  - Denoted by $*$ or simply $\bowtie$ without any predicate
  - Changes schema by retaining *only one copy* of common attribute
  - $r * s = r \bowtie s = r \bowtie_{r.A=s.A} s$
- Applying $\bowtie$ on

A	B
1	1
1	2
2	1

and

A	C
1	2
2	3

returns

Join in Relational Algebra

- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- Join is too common a query to not have its own operator
- Schema is same as $r \times s$ but (potentially) less number of tuples
- The above form is the most general, called the theta join
- Equality join: When the join condition only contains equality
  - $r \bowtie_{B=C} s$
- Natural join: If two relations share an attribute (also its *name*), equality join on that common attribute
  - Denoted by $*$ or simply $\bowtie$ without any predicate
  - Changes schema by retaining *only one copy* of common attribute
  - $r * s = r \bowtie s = r \bowtie_{r.A=s.A} s$
- Applying $\bowtie$ on

A	B
1	1
1	2
2	1

and

A	C
1	2
2	3

returns

A	B	C
1	1	2
1	2	2
2	1	3

Join

- Join types: inner join or simply join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

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

Join

- Join types: inner join or simply join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

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

Join

- Join types: inner join or simply join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

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

Join

- Join types: inner join or simply join
- Join conditions: natural, on ⟨ predicate ⟩, using (⟨ attribute list ⟩)
- Examples

```sql
student inner join program on student.roll = program.roll;
student natural join program;
student join program using (roll);
```

- Multiple relations can be joined

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

Variants in SQL

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