CS2102 Database Systems

Slides adapted from Prof. Chan Chee Yong

LECTURE 03

SQL #1

PREVIOUS LECTURE

Relational data model

Relation schema

- Each relation has a definition called a relation schema
  - Schema specifies attributes and data constraints
  - Data constrains include domain constraints
  - Each row in a relation is called a tuple/record

Relations

- A relation is defined as a set of <u>*tuples*</u>

Relational database schema

- A relational database schema consists of a set of <u>*schemas*</u>
- *Relational database schema*
  = *relational schemas* + *data constraints*

Relational database

- A relational database is a collection of <u>*tables*</u>

PREVIOUS LECTURE

Integrity constraints (ICs)

Definitions

- Integrity constraint
  - A condition that *restricts* the data that can be stored in database instance

Types

- Domain constraints
  - Restrict attribute values of relations
- Key constraints
- Foreign key constraints
- Other general constraints

PREVIOUS LECTURE

Key constraints

Superkey

- A superkey is a subset of attributes in a relation that *uniquely identifies* its tuples
  - No two distinct tuples of relation have the same values in all attributes of superkey

Key

- A key is a superkey that satisfies the additional property
  - Not `null` & no *proper subset* of a key is a superkey
  - ➢ *Minimal* subset of attributes that *uniquely identifies* its tuples
- A relation can have *multiple keys*
  - These are called candidate keys
  - One of the candidate keys is then selected as the primary keys

PREVIOUS LECTURE

Foreign key constraints

Foreign key

- A subset of attributes in a relation is a foreign key if it *refers to the primary key of a second relation*
- Foreign key constraints
    - Each foreign key value in referencing relation must either
        - *Appear as primary key* value in referenced relation, *or*
        - Be a `null` value
    - Referencing & referenced relations could be the same relation
    - Also called referential integrity constraints

PREVIOUS LECTURE

Relational algebra

Selection: $\boldsymbol{\sigma_c}$

- $\sigma_c(R)$ selects tuples from relation R that satisfies selection condition c

Projection: $\boldsymbol{\pi_\ell}$

- $\pi_\ell(R)$ projects attributes given by a list ℓ of attributes from relation R

Renaming: $\boldsymbol{\rho_{S(B_1,B_2,\ldots,B_n)}(R)}$

- $\rho_{S(B_1,B_2,\ldots,B_n)}(R)$ renames $R(A_1, A_2, \ldots, A_n)$ to $S(B_1, B_2, \ldots, B_n)$
  - When the attributes are not renamed $\boldsymbol{\rho_S(R)}$
  - When the table is not renamed $\boldsymbol{\rho_{(B_1,B_2,\ldots,B_n)}(R)}$

PREVIOUS LECTURE

Relational algebra

Union: $\boldsymbol{R \cup S}$

- Returns a relation containing all tuples that occur in R, S, or both

Intersection: $\boldsymbol{R \cap S}$

- Returns a relation containing all tuples that occur in both R and S

Set-difference: $\boldsymbol{R - S}$

- Returns a relation containing all tuples that occur in R but not in S

❖ Union (∪), intersection (∩), and set-difference (−) operators require input relations to be union compatible

Cross-product: ×

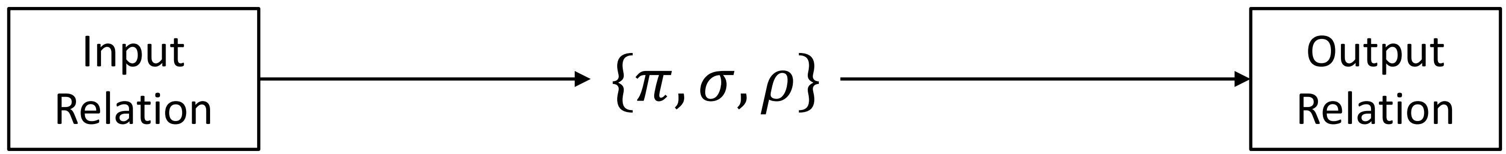
- Consider a relation $R_1(A, B, C)$ and $R_2(X, Y)$
- $R_1 \times R_2$ returns a relation with schema (A, B, C, X, Y) defined as follows:
  - $R_1 \times R_2 = \{(a, b, c, x, y) \mid (a, b, c) \in R_1, (x, y) \in R_2\}$
- Also known as cartesian product

PREVIOUS LECTURE

Closure properties

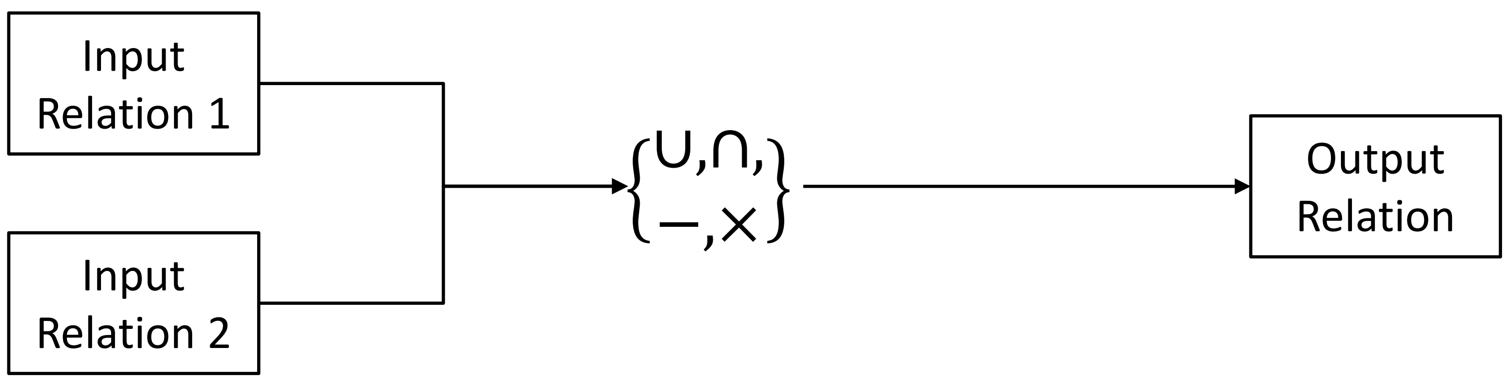
Closure of relation under unary operators *(as diagrams)*

- Unary operator takes in a *<u>relation as input</u>* and gives a *<u>relation as output</u>*



Closure of relation under binary operators *(as diagrams)*

- Binary operator takes in two *<u>relations as inputs</u>* and gives a *<u>relation as output</u>*



- Structured Query Language
  - Null values
  - Create/drop table
  - Table modification
  - Constraint checking
- Queries
  - Simple queries

Overview

- Structured Query Language
  - Null values
  - Create/drop table
  - Table modification
  - Constraint checking
- Queries
  - Simple queries

Structured Query Language

SELF-READ

Structured Query Language

Introduction

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

SELF-READ

Structured Query Language

Using SQL

- **Directly write SQL statements**
  - *Command line interface*
    - PostgreSQL's psql
  - *Graphical user interface*
    - PostgreSQL's pgAdmin
- **Included SQL in application programs**
  - *Statement-level interface (SLI)*
    - Embdded SQL
    - Dynamic SQL
  - *Call-level interface (CLI)*
    - JDBC (Java DataBase Connectivity)
    - ODBC (Open DataBase Connectivity)

SELF-READ

<u>S</u>tructured <u>Q</u>uery <u>L</u>anguage

About SQL

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

Null values

Three-valued logic system

- TRUE
- FALSE
- UNKNOWN

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

Null values

Three-valued logic system

- `TRUE`
- `FALSE`
- `UNKNOWN`

Operation

- Result of comparison operation involving `null` value is *unknown*
- Result of arithmetic operation involving `null` value is *null*

Example: assume that the value of x is `null`

- `x < 100` →
- `x = null` →
- `x <> null` →
- `x + 20` →

Null values

Three-valued logic system

- `TRUE`
- `FALSE`
- `UNKNOWN`

Check

- How to check if a value is *equal* to `null`?
- Use the `IS NULL` comparison predicate

x	***x IS NULL***	***x IS NOT NULL***
`null`		
`non-null`		

Null values

Three-valued logic system

- `TRUE`
- `FALSE`
- UNKNOWN

Check

- How to treat `null` values as ordinary values for comparison?
- Use the `IS DISTINCT FROM` comparison predicate

x	*y*	*x IS DISTINCT FROM y*
null		
null		
non-null		
non-null		

SQL syntax

Comments

- Comments in SQL are preceded by two hyphens
- `-- this is a single-line comment`

C-style comments

- SQL also supports C-style comments
- ```
  /* this is also a comment
     and can be multi-line */
  ```

Grammar

- Keywords are in `UPPERCASE`, variables are in **`lowercase`**
- Optional components are in `[ square brackets ]`
- Choices are separated by `|`
- Non-optional choices are in `{ curly brackets }`
- Possibly infinite repetitions are denoted by `...`
  ```

Create/drop table

Create table syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type
      [ column_constraints [ ... ] ]
    | table_constraints }
  [, ...]
] );
```

- Reference: https://www.postgresql.org/docs/11/sql-createtable.html

Drop table syntax

```
DROP TABLE [ IF EXISTS ] table_name
```

- Reference: https://www.postgresql.org/docs/11/sql-droptable.html

Create/drop table

Examples

```
CREATE TABLE Students (
    -- column1  data_type1
    -- column2  data_type2
    -- column3  data_type3
    -- etc
  );


DROP TABLE Students;
```

Create/drop table

Specifying constraints

- Domain constraints

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type
      [ column_constraints [ ... ] ]
    | table_constraints }
  [, ...]
] )
```

Create/drop table

Specifying constraints

- Domain constraints
  - Built-in data types
  - Domain of each includes a special value `null`
  - Reference: https://www.postgresql.org/docs/11/datatype.html

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

Create/drop table

Examples

```
CREATE TABLE Students (
  studentID    integer,
  name         varchar(100),
  birthDate    date


);
```

students

studentID	*name*	*birthDate*
3118	Alice	1999-12-25
3118	Trudy	1999-12-25
1423	Bob	2000-05-27
5609	Carol	1999-06-11

Create/drop table

Specifying constraints

- Key constraints

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type
      [ column_constraints [ ... ] ]
    | table_constraints }
  [, ...]
] )
```

Create/drop table

Specifying constraints

- Key constraints
  - *Recap:*
    - Superkey *uniquely identifies its tuples*
      - *No two distinct tuples of relations have the same values in all attributes of superkey*
    - Key *minimal superkey*
    - Candidate key *if there are multiple keys*
    - Primary key *one is selected as primary key*
      - Keyword: `PRIMARY KEY`

Create/drop table

Examples

```
CREATE TABLE Students ( -- column constraints
  studentID     integer PRIMARY KEY, -- one PK
  name          varchar(100),
  birthDate     date


);
```

Students

studentID	***name***	***birthDate***
3118	Alice	1999-12-25
3118	Trudy	1999-12-25
1423	Bob	2000-05-27
5609	Carol	1999-06-11

Students

studentID	***name***	***birthDate***
3118	Alice	1999-12-25
1423	Bob	2000-05-27
5609	Carol	1999-06-11

Create/drop table

Examples

```
CREATE TABLE Enrolls ( -- table constraints
  sid    integer,
  cid    integer,
  grade  char(2),
  PRIMARY KEY(sid, cid) -- one or more PK
);
```

Enrolls

sid	*cid*	*grade*
3118	112	3.8
1423	101	4.0
1423	311	3.8
3118	112	4.0

Enrolls

sid	*cid*	*grade*
1423	101	4.0
1423	311	3.8
3118	112	4.0

Create/drop table

Specifying constraints

- Foreign key constraints

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type
      [ column_constraints [ ... ] ]
    | table_constraints }
  [, ...]
] )
```

Create/drop table

Specifying constraints

- Foreign key constraints
  - Recap:
    - Foreign key *refers to primary key of second relation*
      - Each foreign key value in referencing relation must either
        - *Appear as primary key* value in referenced relation, *or*
        - Be a `null` value
      - Referencing & referenced relations could be the same relation
      - Also called referential integrity constraints
      - Keyword: `FOREIGN KEY` and `REFERENCES`

Create/drop table

Examples

```
CREATE TABLE Enrolls (
  sid    integer REFERENCES Students(studentID),
  cid    integer, -- assume students and courses
  grade  char(2), -- table are defined
  FOREIGN KEY(cid) REFERENCES Courses(courseID)
);
```

Enrolls

sid	*cid*	*grade*
1423	101	4.0
1423	311	3.8
3118	112	4.0
null	313	4.5
5609	null	5.0
null	null	0.0
1111	312	1.2

Students

studentID	..	..
3118	..	..
1423	..	..
5609	..	..

Courses

courseID	..	..
101	..	..
312	..	..
112	..	..

Create/drop table

Examples

```
CREATE TABLE Enrolls (
  sid    integer, -- assume Student_Course table
  cid    integer, -- is defined
  FOREIGN KEY(sid,cid) REFERENCES
    Student_Course(studentID, courseID)
);
```

Enrolls

sid	*cid*	*grade*
1423	101	4.0
1423	311	3.8
3118	112	4.0
null	313	4.5
5609	null	5.0
null	null	0.0
1111	312	1.2

Student_Course

studentID	*courseID*
3118	112
1423	101
1423	312
5609	112

Constraint checking

Foreign key constraint violation

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

```
FOREIGN KEY(...) REFERENCES ...
    [ ON DELETE action ] [ ON UPDATE action ]
```

Students

studentID	..	..
3118	..	..
1423	..	..
5609	..	..

Enrolls

sid	*cid*	*grade*
1423	101	4.0
3118	112	4.0
5609	312	1.2

Constraint checking

Foreign key constraint violation

```
FOREIGN KEY(...) REFERENCES ...
    [ ON DELETE action ] [ ON UPDATE action ]
```

Actions

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

Constraint checking

Foreign key constraint violation

```
FOREIGN KEY(...) REFERENCES ...
    ON DELETE SET NULL    ON UPDATE CASCADE

-- delete 3118

-- change 1423 to 1444
```

Students

studentID	..	..
1444	..	..
5609	..	..

Enrolls

sid	*cid*	*grade*
1444	101	4.0
null	112	4.0
5609	312	1.2

Create/drop table

Specifying constraints

- Other general constraints

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type
      [ column_constraints [ ... ] ]
    | table_constraints }
  [, ...]
] )
```

Create/drop table

Specifying constraints

- Other general constraints
  - Not-null constraints *keyword*: `NOT NULL`
  - Unique constraints *keyword*: `UNIQUE`
    - Check any two records $x, y \in \text{table}$ such that

$$x.\text{column} <> y.\text{column}$$

  - General constraints *keyword*: `CHECK`
  - Constraint is violated if it evaluates to `FALSE`

- Default values
  - Specify default values *keyword*: `DEFAULT`

Create/drop table

Examples: not-null

```
CREATE TABLE Students (
  studentID    integer,
  name         varchar(100) NOT NULL,
  birthDate    date
);

CREATE TABLE Students (
  studentID    integer,
  name         varchar(100),
  birthDate    date,
  CHECK (name IS NOT NULL)
);
```

Create/drop table

Examples: unique

```
CREATE TABLE Students (
  studentID    integer,
  name         varchar(100) UNIQUE,
  birthDate    date
);
```

Students

studentID	***name***	***birthDate***
3118	Alice	1999-12-25
3119	Alice	1999-12-25
1423	Bob	2000-05-27
5609	Carol	1999-06-11

Create/drop table

Examples: unique

```
CREATE TABLE Students (
   studentID     integer,
   name          varchar(100),
   birthDate     date,
   UNIQUE (studentID, name)
);
```

Students

studentID	***name***	***birthDate***
3118	Alice	1999-12-25
3119	Alice	1999-12-25
1423	Bob	2000-05-27
1423	Bob	1999-06-11

Create/drop table

Examples: general

```
CREATE TABLE Students (
  studentID     integer,
  name          varchar(100),
  birthDate     date,
  CHECK (studentID > 2000)
);
```

Students

<u>studentID</u>	***name***	***birthDate***
3118	Alice	1999-12-25
3119	Alice	1999-12-25
1423	Bob	2000-05-27
5609	Carol	1999-06-11

Create/drop table

Examples: default values

```
CREATE TABLE Students (
  studentID     integer,
  name          varchar(100) DEFAULT ‘John Doe’,
  birthDate     date
);
```

Create/drop table

Assertions

- Complex constraints
  - Multi-table constraints
  - Example:
    - Every student in `students` table *must be enrolled in at least two and at most five courses* in `enrolls` table
  - Limitation
    - *Create assertion statement is not implemented in many DBMSs*
    - We can use `triggers` to enforce complex constraints
      - Lecture 06

Table modification

Insert into syntax

```
INSERT INTO table_name
[ ( column_name [, ...] ) ]
VALUES ( { expression | DEFAULT } [, ...] );
```

Delete from syntax

```
DELETE FROM table_name
[ WHERE condition ];
```

Update syntax

```
UPDATE table_name
SET column_name = { expression | DEFAULT }
[ WHERE condition ];
```

Table modification

Example

```
CREATE TABLE Students (
  studentID    integer PRIMARY KEY,
  name         varchar(100),
  dept         varchar(20) DEFAULT ‘CS’
);
INSERT INTO Students VALUES (12345, ‘Alice’, ‘Eng’);
INSERT INTO Students (studentID) VALUES (23456);
```

~~INSERT INTO Students (studentID) VALUES (12345);~~

Students

<u>studentID</u>	***name***	***dept***
12345	Alice	Eng
23456	null	CS

Table modification

Example

```
INSERT INTO Students VALUES (34567, ‘Bob’, ‘Eng’);
INSERT INTO Students (dept, name, studentID)
VALUES (‘Maths’, ‘Carol’, 45678);
DELETE FROM Students WHERE dept=‘Eng’;
DELETE FROM Students; -- remove all
INSERT INTO Students VALUES (12345, ‘Alice’, ‘Eng’);
INSERT INTO Students (studentID) VALUES (23456);
```

Students

studentID	*name*	*dept*
12345	Alice	Eng
23456	null	CS
~~34567~~	~~Bob~~	~~Eng~~
~~45678~~	~~Carol~~	~~Maths~~

Table modification

Example

```
UPDATE Students SET name = ‘Bob’ WHERE dept = ‘CS’;

UPDATE Students SET studentID = studentID + 1;

-- update all
```

students

<u>studentID</u>	***name***	***dept***
12346	Alice	Eng
23457	Bob	CS

Schema modification

Alter table

- Add/remove/modify columns
  - `ALTER TABLE Students ALTER COLUMN dept DROP DEFAULT;`
  - `ALTER TABLE Students DROP COLUMN dept;`
  - `ALTER TABLE Students ADD COLUMN faculty varchar(20);`
  - etc
- Add/remove constraints
- etc

- Structured Query Language
  - Null values
  - Create/drop table
  - Table modification
  - Constraint checking
- Queries
  - Simple queries

Queries

Simple queries

Basic syntax

- Basic form of SQL query consists of *three clauses*

```
SELECT [ DISTINCT ]  select_list  -- select clause
FROM                 from_list    -- from clause
[ WHERE              condition ]  -- where clause
```

- `select_list` specifies columns to be included in output
- `from_list` specifies list of relations
- `condition` specifies conditions on relations

- **Output:** relation generated from `from_list` containing attributes based on `select_list` that satisfies `condition`
  - Output relation could contain duplicate record if `DISTINCT` is not used in the `SELECT` clause

Simple queries

Basic syntax

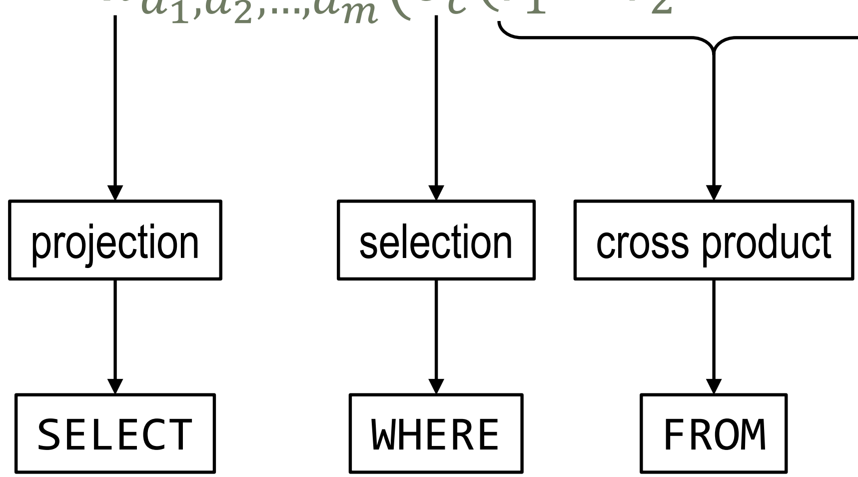
- Basic form of SQL query consists of *three clauses*

$\texttt{SELECT} \quad \texttt{DISTINCT} \quad \boldsymbol{a_1, a_2, \dots, a_m}$ -- select clause

$\texttt{FROM} \quad \boldsymbol{r_1, r_2, \dots, r_n}$ -- from clause

$\texttt{WHERE} \quad \boldsymbol{c}$ -- where clause

- **Relational algebra form**
  - $\pi_{a_1,a_2,\dots,a_m}\left(\sigma_c\left(\underbrace{r_1 \times r_2 \times \cdots \times r_n}\right)\right)$



What about renaming? ρ

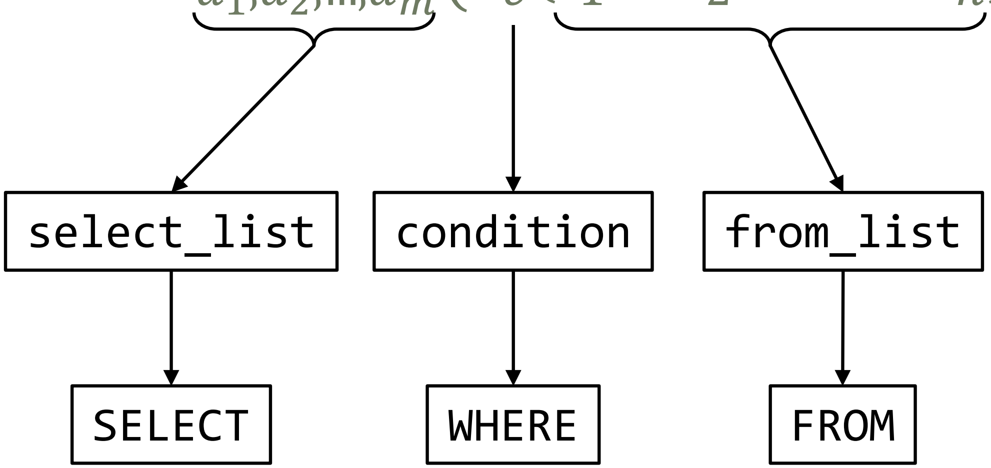
Simple queries

Basic syntax

- Basic form of SQL query consists of <u>*three clauses*</u>

```
SELECT   DISTINCT    a1, a2, ..., am   -- select clause
FROM                 r1, r2, ..., rn   -- from clause
  WHERE              c                 -- where clause
```

- **Relational algebra form**
  - $\pi_{a_1, a_2, \ldots, a_m}\left(\sigma_c\left(r_1 \times r_2 \times \cdots \times r_n\right)\right)$



Simple queries

Example

- Find the names of restaurants, the pizzas that they sell, and their prices, where the price is under $20
- Deconstruct problem
  - Find what?
  - From which relation?
  - What condition?
- Construct query

Sells

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

Output

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

Simple queries

Example

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

```
SELECT rname, pizza, price
FROM   Sells
WHERE  price < 20;
```

- Alternative query

```
SELECT *
FROM   Sells
WHERE  price < 20;
```

Output

__rname__	*__pizza__*	*price*
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Pizza King	Diavola	17

Sells

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

Simple queries

Example

- Find all restaurants, the pizzas that they sell, and their prices, where (1) either the price is under $20 or the pizza is "Marinara", and (2) the pizza is not "Diavola"
- Construct query

```
SELECT rname, pizza, price -- or simply *
FROM   Sells
WHERE  (price < 20 OR pizza = 'Marinara')
  AND  pizza <> 'Diavola';
```

Sells

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

Output

rname	*pizza*	*price*
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Mamma's Place	Marinara	22

Simple queries

Removing duplicates

- `q1: SELECT a, c FROM r;`
- `q2: SELECT DISTINCT a, c FROM r;`

r

a	b	c
10	1	2
10	7	2
20	3	Null
20	9	Null
30	3	2
30	5	9

q1

a	c
10	2
10	2
20	Null
20	Null
30	2
30	9

q2

a	c
10	2
20	Null
30	2
30	9

- Two tuples $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$ are considered to be distinct if the following evaluates to `TRUE`:
  - $(a_1$ `IS DISTINCT FROM` $b_1)$ `or`
    $(a_2$ `IS DISTINCT FROM` $b_2)$ `or`
    `... or`
    $(a_n$ `IS DISTINCT FROM` $b_n)$

Simple queries

Renaming column

- q: SELECT item, price * qty AS cost FROM Orders;

Orders

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

q

item	Cost
A	250.00
B	400.00
C	750.00

Renaming values

- q: SELECT ‘Price of ’ || pizza || ‘ is ’ ||
  round(price / 1.3) || ‘ USD’ AS menu
  FROM Sells WHERE rname = ‘Corleone Corner’;

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Pizza King	Hawaiian	21

q

menu
Price of Diavola is 18 USD
Price of Hawaiian is 19 USD
Price of Margherita is 15 USD

Summary

- SQL is the standard query language for relational DBMS
  - Table creation `CREATE TABLE table_name`
  - Table removal `DROP TABLE table_name`
  - Modification `ALTER TABLE table_name`
    - Insert `INSERT INTO table_name VALUES ( .. )`
    - Delete `DELETE FROM table_name WHERE ..`
    - Update `UPDATE table_name SET .. WHERE ..`
  - Queries

```
SELECT DISTINCT   a_1, a_2, ..., a_m
FROM              r_1, r_2, ..., r_n
WHERE             c
```

$$\text{SELECT DISTINCT } a_1, a_2, \dots, a_m \quad \text{FROM } r_1, r_2, \dots, r_n \quad \text{WHERE } c$$

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