CS2102 Database Systems

STRUCTURED QUERY LANGUAGE

Structured Query Language

- Developed in IBM Research Laboratory in the 1970s
- Industry standard for relational databases (SQL92 is an ANSI/ISO standard)

Structured Query Language

- Data Definition Language
  - Create, delete, modify schemas
  - Define integrity constraints, views, triggers
- Data Manipulation Language
  - Ask queries
  - Insert, delete, modify tuples
- Database Control Language
  - Define access rights, concurrency control

SQL DDL, creation (simple)

```
CREATE TABLE relation-name
        (attribute-name domain
        [, attribute-name domain ]*)
```

SQL DDL, creation example

```
CREATE TABLE Branch
    ( name        VARCHAR(10),
      city        VARCHAR (20),
      director    VARCHAR (20),
      assets      NUMERIC)
```

Branch

name	city	director	assets

SQL DDL, Domains

- Character
  - CHAR(n)
  - VARCHAR(n) (Oracle)
- Bit (SQL-92)
- Numeric
- Date and Time
- Temporal Interval

SQL DDL, creation

```
CREATE TABLE relation-name
        (attribute-name domain [DEFAULT expr]
         [column_constraint]*   e.g. cannot have null values
         [,attribute-name domain [DEFAULT expr]
         [column_constraint]* ]*
         [,table_constraint]*);
```

SQL DDL, creation example

```
CREATE TABLE Employee (
        emp_name      VARCHAR(24)  PRIMARY KEY,
        address       VARCHAR(36)  DEFAULT 'company address'
)
```

if user does not specify, then the system will use the company address as the default value

SQL DDL, Integrity Constraints

only need to check one column

- Column-level or Table-level

check if there is any modification to a row, multiple columns involved in the check too

- 5 categories
  - PRIMARY KEY
  - REFERENCES
    - Foreign key or referential constraint
  - UNIQUE
  - NOT NULL
  - CHECK
    - Generalized dependences expressed as valid condition in WHERE clause

note: unique not null is a candidate key

SQL DDL, creation example

```
CREATE TABLE Branch
        ( name          VARCHAR(10) ,
          city          VARCHAR(20) DEFAULT 'Singapore',
          director      VARCHAR(20) UNIQUE,
          assets        NUMERIC CHECK (assets > 0),
          PRIMARY KEY (name, city))
```

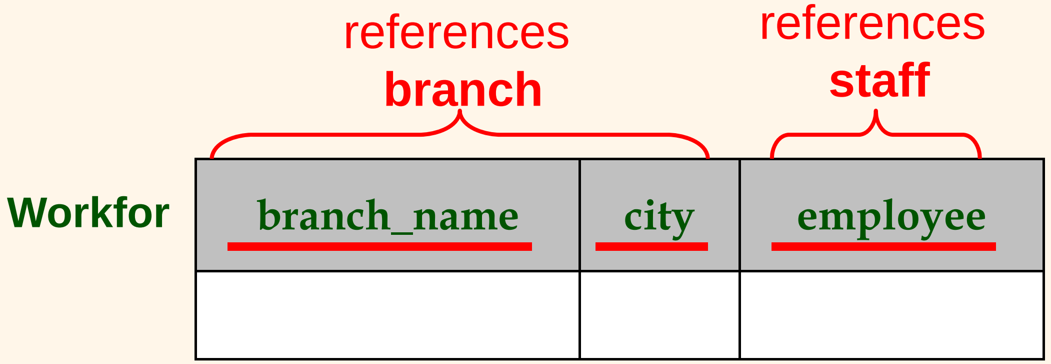
table constraint

Branch

name	city	director	assets

SQL DDL, creation example

```
CREATE TABLE Workfor
        ( branch_name   VARCHAR(10) ,
          city   VARCHAR(20)   DEFAULT 'Singapore',
          employee   VARCHAR(20)  REFERENCES Staff(name),
          FOREIGN KEY (branch_name, city)
                REFERENCES  Branch(name, city),
          PRIMARY KEY (branch_name, city, employee)
        )
```



SQL DDL, deletion

DROP TABLE *relation_name*

SQL DDL, deletion example

DROP TABLE *Branch*

SQL DDL, alteration

ALTER TABLE *relation_name* ADD *Att Domain*

ALTER TABLE *relation_name* DROP *Att*

SQL DDL, alteration example

ALTER TABLE *Branch* ADD *zip VARCHAR(6)*

Branch

name	**city**	**director**	**assets**	**zip**

ALTER TABLE *Branch* DROP *zip*

Branch

name	**city**	**director**	**assets**

SQL DML, insertion

INSERT INTO *relation_name [(Att [,Att]*)]*
        VALUES *(value [,value]*)*

INSERT INTO *relation_name [(Att [,Att]*)]* ***query***

SQL DML, insertion example

need not follow the order of attributes specified in table creation,
however, this order indicates the order of values you are about to insert

INSERT INTO *Branch (name, director, city, assets)*
VALUES (*'Clementi', 'Ng Wee Hiong',*
'Singapore', 3000000)

Branch

name	city	director	assets
Clementi	Singapore	Ng Wee Hiong	3000000

SQL DML, insertion example

you can even insert values from another table

INSERT INTO *JohorDirector* *(assume table was created)*

SELECT *director*

FROM *Branch* WHERE *city* = *'Johor Barhu'*

Branch

name	city	director	assets
Clementi	Singapore	Ng Wee Hiong	3000000
F_branch	Johor Barhu	John	1500000
S_branch	Johor Barhu	George	1200000

JohorDirector

director
John
George

SQL DML, deletion

DELETE FROM *relation_name*
[WHERE *qualification*]

SQL DML, deletion example

DELETE FROM *Branch*

WHERE *city* = *'Jakarta'* and *assets* < *1000000*

Branch

name	city	director	assets
Clementi	Singapore	Ng Wee Hiong	3000000
F_branch	Johor Barhu	John	1500000
S_branch	Johor Barhu	George	1200000
~~Branch_one~~	~~Jakarta~~	~~Bo Lee~~	~~80000~~
Monas	Jakarta	Agus Arianto	4000000

SQL DML, update

```
UPDATE relation_name
SET att = expr
[WHERE qualification]
```

SQL DML, update example

Branch

name	city	director	assets
Clementi	Singapore	Ng Wee Hiong	3000000
F_branch	Johor Bahru	John	1500000
KL_branch	Kuala Lumpur	Yu Fei	1000000

UPDATE *Branch*

SET *assets* = *assets * 1.5*

WHERE *city* = *'Kuala Lumpur'*

Branch

name	city	director	assets
Clementi	Singapore	Ng Wee Hiong	3000000
F_branch	Johor Bahru	John	1500000
KL_branch	Kuala Lumpur	Yu Fei	1500000

SQL DML, simple query

- Basic form of SQL query has 3 clauses:

```
SELECT  [DISTINCT] attribute-list
FROM        relation-list
[WHERE      qualification]
```

  - relation-list: specifies list of relations to compute cross product
  - attribute-list: specifies columns to be projected for output table
  - qualification: specifies selection conditions
  - Output relation can contain duplicates if DISTINCT is not used

SQL DML, simple query example

```
SELECT    *
FROM   Workfor
```

Workfor

branch_name	**city**	**employee**
Clementi	Kuala Lumpur	Yu Fei
Clementi	Singapore	Ng Wee Hiong
Clementi	Singapore	Peter Ho
Clementi	Singapore	Jean Do
Monas	Jakarta	Agus Arianto
Monas	Jakarta	Reza Santi

SQL DML, simple query example

Branch

name	city	director	assets
Branch_one	Jakarta	Bo Lee	80000
Clementi	Singapore	Ng Wee Hiong	3000000
F_branch	Johor Barhu	John	1500000
KL_branch	Kuala Lumpur	Yu Fei	1000000
Monas	Jakarta	Agus Arianto	4000000
S_branch	Johor Barhu	George	1200000

SELECT *name, city*
FROM *Branch*

name	city
Branch_one	Jakarta
Clementi	Singapore
F_branch	Johor Barhu
KL_branch	Kuala Lumpur
Monas	Jakarta
S_branch	Johor Barhu

SQL DML, simple query example

name	city	director	assets
Branch_one	Jakarta	Bo Lee	80000
Clementi	Singapore	Ng Wee Hiong	3000000
F_branch	Johor Barhu	John	1500000
KL_branch	Kuala Lumpur	Yu Fei	1000000
Monas	Jakarta	Agus Arianto	4000000
S_branch	Johor Barhu	George	1200000

```
SELECT    name
FROM      Branch
WHERE     city = 'Jakarta'
AND  assets <1000000
```

name
Branch_one

Select – Project – Join Query

Workfor

branch_name	city	employee
Clementi	Singapore	Ng Wee Hiong
Clementi	Singapore	Peter Ho
Clementi	Singapore	Jean Do
Monas	Jakarta	Agus Arianto
Monas	Jakarta	Reza Santi

Branch

name	city	director	assets
Clementi	Singapore	Ng Wee Hiong	3000000
Monas	Jakarta	Agus Arianto	4000000

SELECT DISTINCT *employee, director*

FROM *Branch, Workfor*

WHERE *name* = *branch_name*

AND *branch.city* = *workfor.city*

since the attribute name 'city' is not unique, you need to let the system know which table's 'city' attribute you are referring to

employee	director
Agus Arianto	Agus Arianto
Jean Do	Ng Wee Hiong
Ng Wee Hiong	Ng Wee Hiong
Peter Ho	Ng Wee Hiong
Reza Santi	Agus Arianto

cross product, followed by elimination

Branch				Workfor		
name	**city**	**director**	**assets**	**branch_name**	**city**	**employee**
Clementi	Singapore	Ng Wee Hiong	3000000	Clementi	Singapore	Ng Wee Hiong
Clementi	Singapore	Ng Wee Hiong	3000000	Clementi	Singapore	Peter Ho
Clementi	Singapore	Ng Wee Hiong	3000000	Clementi	Singapore	Jean Do
~~Clementi~~	~~Singapore~~	~~Ng Wee~~ Hiong	~~3000000~~	~~Monas~~	~~Jakarta~~	~~Agus Arianto~~
~~Clementi~~	~~Singapore~~	~~Ng Wee~~ Hiong	~~3000000~~	~~Monas~~	~~Jakarta~~	~~Reza Santi~~
~~Monas~~	~~Jakarta~~	~~Agus Arianto~~	~~4000000~~	~~Clementi~~	~~Singapore~~	~~Ng Wee~~ Hiong
~~Monas~~	~~Jakarta~~	~~Agus Arianto~~	~~4000000~~	~~Clementi~~	~~Singapore~~	~~Peter Ho~~
~~Monas~~	~~Jakarta~~	~~Agus Arianto~~	~~4000000~~	~~Clementi~~	~~Singapore~~	~~Jean Do~~
Monas	Jakarta	Agus Arianto	4000000	Monas	Jakarta	Agus Arianto
Monas	Jakarta	Agus Arianto	4000000	Monas	Jakarta	Reza Santi

Bag Semantics (**not** *SET*)

SELECT *branch_name*
FROM *Workfor*

the system will just take out every row, not keeping duplicates in mind

branch_name
Clementi
Clementi
Clementi
Monas
Monas

SELECT **DISTINCT** *branch_name*
FROM *Workfor*

elimintates duplicates

branch_name
Clementi
Monas

However… List Semantics

```
SELECT      name, city
FROM        Branch
ORDER BY    name ASC,  city DESC
```

you can sort the results by name in ascending order,
with city in descending order

Arithmetic in SQL

can be done in the select and where clause

Branch

name	city	director	assets
Clementi	Singapore	Ng Wee Hiong	3000000
Monas	Jakarta	Agus Arianto	4000000

SELECT *name, city, assets*1.7 as assets_USD*
FROM *Branch*

name	city	assets_USD
Clementi	Singapore	5100000
Monas	Jakarta	6800000

SELECT *name, city*
FROM *Branch*
WHERE *assets*1.7 < 1700000*

Dealing with Ambiguity

SELECT DISTINCT ***Workfor.***_employee_, ***Branch.***_director_
FROM _Branch, Workfor_
WHERE ***Branch.***_name_ = ***Workfor.***_branch_name_
AND ***Branch.***_city_ = ***Workfor.***_city_

❖ Qualify *city* with a prefix *Branch.* or *Workfor.* to disambiguate the common attribute name *city* between relations *Branch* and *Workfor*

Renaming Tables

SELECT DISTINCT *employee, director*
FROM Branch ***B***, *Workfor* ***W***
WHERE *name = branch_name*
AND ***B.****city =* ***W.****city*

employee	director
Agus Arianto	Agus Arianto
Jean Do	Ng Wee Hiong
Ng Wee Hiong	Ng Wee Hiong
Peter Ho	Ng Wee Hiong
Reza Santi	Agus Arianto

Renaming Tables

Find pairs of branches (b1, b2) where b1 has more assets than b2

```
SELECT B1.name, B2.name
FROM   Branches B1 , Branches B2
WHERE  B1.assets > B2.assets
```

- ***B1*** and ***B2*** are called <u>range variable</u> (or tuple variables).
- Range variables are used to refer to tables in FROM clause
- Column names can be prefixed by a range variable
- Range variables are convenient and useful when same table name appears multiple times in FROM clause

Conceptual Evaluation of Queries

- Output of an SQL query is a relation: a multiset of rows
- Semantics of a basic SQL query can be explained using the following conceptual evaluation:

> **SELECT [DISTINCT]** attribute-list
> **FROM** relation-list
> [**WHERE** qualification]

- Compute cross-product of the tables in the **relation-list**.
- Delete rows in the cross-product that fail the **qualification** conditions
- Delete columns that do not appear in the **attribute-list**
- If **DISTINCT** is specified, eliminate duplicate rows.

Expressions in SELECT Clause

- Renaming column names using AS keyword

  SELECT title **AS** MovieTitle, rating **AS** reviewScore
  FROM Movies

- Numeric or string constants

  SELECT 'Rating for' | | title | | ' is ' | | rating
  FROM Movies

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Expressions in SELECT Clause

- Arithmetic expressions

```
SELECT title, (rating + 0.2) * 10
FROM   Movies
```

- Aggregation operators: COUNT, SUM, AVG, MIN, MAX

```
SELECT COUNT(title), AVG(rating)
FROM   Movies
```

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Conditions in WHERE Clause

- Qualification in WHERE clause is a boolean combination of conditions
- Each condition could be
  - basic comparison condition

    expression op expression
  - set comparison condition
- *op* is a comparison operators: = , <> , < , > , <= , >=
- *expression* is a column name, a constant, or an arithmetic/string expression

Conditions in WHERE Clause

- ❖ Conditions are combined using logical connectors: AND, OR, NOT

```
SELECT title
FROM   Movies
WHERE ( (director = 'Coen') OR (rating * 10 < 70) )
AND     NOT (myear = 1999)
```

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Conditions in WHERE Clause

- SQL provides support for pattern matching using the LIKE operator
- Find movies whose title begins with *W* and ends with *S* and has at least three characters

```
SELECT title
FROM   Movies
WHERE title LIKE 'W_%S'
```

- Symbol % stands for 0 or more arbitrary characters
- Symbol _ stands for a single arbitrary character

Set Comparison Operations

- Set comparison operations in WHERE clause:
  - *v* IN Q is *true* iff value *v* is in the set returned by Q
  - *v* NOT IN Q is *true* iff value *v* is not in the set returned by Q
  - EXISTS Q is *true* iff the result of Q is non-empty
  - NOT EXISTS Q is *true* iff the result of Q is empty
  - UNIQUE Q is *true* iff the result of Q has no duplicates
  - *v* op ANY Q is *true* iff there exists some *v*' in the result of Q such that *v* op *v*' is *true*
  - *v* op ALL Q is *true* iff for each *v*' in the result of Q, *v* op *v*' is *true*
  - op $\in$ {= , <> , < , <= , > , >=}
- Q is called a subquery

Nested Queries

- A nested query is a query containing some subquery
- A subquery in a nested query is also called an inner query that is contained in an outer query
- A subquery returns either a constant or a relation
- A subquery can be used in
  - WHERE clause
  - FROM clause
  - HAVING clause

Nested Queries

Find all employees who work in a Singaporean branch

```
SELECT employee
FROM work_for
WHERE branch_name IN (
        SELECT name
        FROM branch
        WHERE  city= 'Singapore')
```

= ANY

Nested Queries

Find all employees who ***do not*** work in a Singaporean branch

```
SELECT employee                              NOT IN
FROM work_for
WHERE branch_name <> ALL (
        SELECT name
        FROM branch
        WHERE  city= 'Singapore')
```

Nested Queries

Find the cities where the average assets of their branches are larger than the global average.

```
SELECT city
FROM branch b1
GROUP BY city
HAVING AVG(b1.assets) > (
        SELECT AVG(b2.assets)
        FROM branch b2)
```

Example Database

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Actors

actor	ayear
Cage	1964
Hanks	1956
Maguire	1975
McDormand	1957

Acts

actor	title
Cage	Raising Arizona
Maguire	Spiderman
Maguire	Wonder Boys
McDormand	Fargo
McDormand	Raising Arizona
McDormand	Wonder Boys

Directors

director	dyear
Coen	1954
Hanson	1945
Raimi	1959

Nested Queries - Example

Find actors who have acted in some movie made before 2000

```
SELECT DISTINCT A.actor
FROM        Acts A
WHERE       A.title IN ( SELECT   M.title
                         FROM     Movies M
                         WHERE    M.myear < 2000 )
```

Is the above query equivalent to the following query?

```
SELECT DISTINCT A.actor
FROM        Acts A, Movies M
WHERE       A.title = M.title
AND         M.year < 2000
```

yes

Nested Queries - Example

Find movies made after 1997 without the actor Maguire

```
SELECT      M.title
FROM        Movies M
WHERE       M.year > 1997
AND         M.title  NOT IN ( SELECT   A.title
                              FROM     Acts A
                              WHERE    A.actor = 'Maguire' )
```

Is the above query equivalent to the following query?

```
SELECT DISTINCT M.title
FROM        Movies M, Acts A
WHERE       M.year > 1997
AND         M.title = A.title
AND         A.actor <> 'Maguire'
```

yes

Nested Queries - Example

Find movies that are rated higher than <u>some</u> Coen's movie

```
SELECT    M.title
FROM      Movies M
WHERE     M.rating > ANY ( SELECT   N.rating
                           FROM     Movies N
                           WHERE    N.director = 'Coen' )
```

Is the above query equivalent to the following query?

```
SELECT    M.title
FROM      Movies M, Movies N
WHERE     N.director = 'Coen'
AND       M.rating > N.rating
```

yes

Nested Queries - Example

Find movies that are rated higher than all of Coen’s movies

```
SELECT     M.title
FROM       Movies M
WHERE      M.rating > ALL ( SELECT   N.rating
                            FROM     Movies N
                            WHERE    N.director = ‘Coen’ )
```

Nested Queries - Example

- A subquery can be nested within another subquery

  Find directors who have made some movie before 2000 with Cage

```
SELECT  D.director
FROM    Directors D
WHERE   D.director  IN ( SELECT M.director
                         FROM   Movies M
                         WHERE  M.myear < 2000
                         AND    M.title  IN (  SELECT A.title
                                               FROM Acts A
                                               WHERE A.actor = ‘Cage’)
```

Correlated Nested Queries

- A correlated nested query is a nested query where there is a subquery that is dependent on the tuple referenced in its outer query

Find movies with rating higher than the average rating of the director's movies

```
SELECT  M.title
FROM    Movies  M
WHERE   M.rating  >  ( SELECT AVG (N.rating)
                       FROM    Movies  N
                       WHERE   N.director = M.director )
```

Correlated Nested Queries

Find directors who have made some movie before 2000 with Cage

```
SELECT       D.director
FROM         Directors D
WHERE   EXISTS  (  SELECT *
                   FROM     Movies M, Acts A
                   WHERE  M.director = D.director
                   AND      M.myear < 2000
                   AND      M.title = A.title
                   AND      A.actor = ‘Cage’ )
```

Subqueries in FROM Clause

Find actors who have acted in some Coen’s Movie

```
SELECT DISTINCT A.actor
FROM    Acts A,
        ( SELECT  M.title AS title
          FROM     Movies M
          WHERE  M.director = ‘Coen’) AS  C
WHERE  A.title = C.title
```

Nested Queries (Variable Scope)

- A reference to attribute can only be used within the SELECT and WHERE clauses where it is defined or within recursively nested queries

```
SELECT city, AVG (b1.assets), AVG (b2.assets)
FROM branch b1
GROUP BY city
HAVING AVG (b1.assets) > ( SELECT AVG(b2.assets)
                           FROM branch b2)
```

- Above query is wrong

Set Operations

- Q_1 UNION Q_2 = $Q_1 \cup Q_2$

need to be union-compatible (same no. of columns and domain of attributes)

- Q1 INTERSECT Q2 = Q1 $\cap$ Q2
- Q1 EXCEPT Q2 = Q1 - Q2
- UNION, INTERSECT and EXCEPT eliminates duplicates
- UNIONALL, INTERSECTALL and EXCEPTALL preserves duplicates

UNION

Find movies with actor Cage <u>or</u> Maguire

```
SELECT A.title
FROM   Acts A
WHERE A.actor = 'Cage'
UNION
SELECT A.title
FROM   Acts A
WHERE A.actor = 'Maguire'
```

```
SELECT A.title
FROM   Acts A
WHERE A.actor = 'Cage'
OR        A.actor = 'Maguire'
```

INTERSECT

Find movies with actors Cage and Maguire

```
SELECT A.title
FROM   Acts A
WHERE A.actor = 'Cage'
INTERSECT
SELECT A.title
FROM   Acts A
WHERE A.actor = 'Maguire'
```

```
SELECT A.title
FROM   Acts A, Acts B
WHERE  A.actor = 'Cage'
AND    A.title = B.title
AND    B.actor = 'Maguire'
```

EXCEPT

Find movies with actors Cage but not actor Maguire

```
SELECT A.title
FROM   Acts A
WHERE A.actor = 'Cage'
EXCEPT
SELECT A.title
FROM   Acts A
WHERE A.actor = 'Maguire'
```

Aggregate Operators

- Aggregate operators appear only in SELECT & HAVING clauses

must not be used in WHERE

- COUNT ([DISTINCT] A)
  - Number of (unique) values in the A column
- COUNT ([DISTINCT] *)
  - Number of (unique) rows.
- SUM ([DISTINCT] A)
  - Sum of all (unique) values in the A column
- AVG ([DISTINCT] A)
  - Average of all (unique) values in the A column

COUNT Operator

Find the number of actors

```
SELECT COUNT (A.actor)
FROM    Actors A
```

Find the number of rows in Actors

```
SELECT COUNT ( * )
FROM    Actors A
```

Find the number of actors who have appeared in some movie

```
SELECT COUNT (DISTINCT A.actor)
FROM    Acts A
```

AVG, MIN, MAX Operators

Find the average rating of Coen's movies

```
SELECT AVG (M.rating)
FROM   Movies M
WHERE M.director = 'Coen'
```

Find the year of birth of the oldest director

```
SELECT MIN ( D.dyear )
FROM   Directors D
```

Aggregate Operators

Nesting of aggregate operators is not allowed

```
SELECT SUM (AVG (M.rating))
FROM   Movies M
```

The above query is illegal !

Grouping Records

❖ How to compute aggregates for groups of records?

Find the maximum rating of each director's movies

1. Partition records in Movies into groups based on director
2. Compute aggregate for each group
3. Output one record for each group

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Answer

director	maxRating
Coen	8.2
Raimi	7.4
Hanson	7.6

Grouping Records

Find the maximum rating of each director's movies

```
SELECT 'Coen', MAX (M.rating)
FROM   Movies M
WHERE M.director = 'Coen'
UNION
SELECT 'Hanson', MAX (M.rating)
FROM   Movies M
WHERE M.director = 'Hanson'
UNION
SELECT 'Raimi', MAX (M.rating)
FROM   Movies M
WHERE M.director = 'Raimi'
```

this is what happens when you don't use GROUP BY clause

Need to know the directors in Movies to write above query

GROUP BY Clause

```
SELECT  M.director, MAX (M.rating) AS maxRating
FROM    Movies M
GROUP BY M.director
```

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Answer

director	maxRating
Coen	8.2
Raimi	7.4
Hanson	7.6

GROUP BY Clause

Find the number of distinct actors who have worked with each director

SELECT M.director, **COUNT** (**DISTINCT** A.actor) **AS** num
FROM Movies M, Acts A
WHERE M.title = A.title
GROUP BY M.director

combine the 2 tables and fulfil qualification in WHERE clause first, then use GROUP BY on it

Answer

director	num
Coen	2
Raimi	1
Hanson	2

$\sigma_{\text{Movies.title = Acts.title}}$ **Movies x Acts**

M.title	M.director	M.myear	M.rating	A.Actor	A.title
Fargo	Coen	1996	8.2	McDormand	Fargo
Raising Arizona	Coen	1987	7.6	Cage	Raising Arizona
Raising Arizona	Coen	1987	7.6	McDormand	Raising Arizona
Spiderman	Raimi	2002	7.4	Maguire	Spiderman
Wonder Boys	Hanson	2000	7.6	Maguire	Wonder Boys
Wonder Boys	Hanson	2000	7.6	McDormand	Wonder Boys

GROUP BY Clause

- If an aggregate operator appears in the SELECT clause and there is no GROUP BY clause, then the SELECT clause must have only aggregate operations

  Find the name and year of birth of the oldest director

This query is illegal !

```
SELECT  D.director, MIN (D.dyear)
FROM    Directors D
```

This query is correct

```
SELECT  D.director, D.dyear
FROM    Directors D
WHERE   D.dyear = ( SELECT MIN (E.dyear) FROM Directors E )
```

HAVING Clause

sets conditions for GROUP BY clause, thus must be used in conjunction with GROUP BY in order for it to work

- HAVING clause specify selection conditions on groups

  For each director who has made at least two movies, find the maximum rating of his movies

```
SELECT     M.director, MAX (M.rating) AS maxRating
FROM       Movies M
GROUP BY   M.director
HAVING     COUNT (*) > 1
```

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Answer

director	maxRating
Coen	8.2

Display total price of all groups, but only those groups with names that start with 'k':

```
SELECT name, SUM(price)
FROM dept
GROUP BY name
HAVING name LIKE 'k%'
```

HAVING Clause

Find actors who have acted in more movies than the number of movies made by Hanson

```
SELECT     A.actor
FROM       Acts A
GROUP BY   A.actor
HAVING     COUNT (*) > ( SELECT COUNT (M.title)
                         FROM   Movies M
                         WHERE M.director = 'Hanson' )
```

remember, you cannot use aggregate operators in WHERE, so you need to resort to GROUP and HAVING COUNT(*)

Acts

actor	title
Cage	Raising Arizona
Maguire	Spiderman
Maguire	Wonder Boys
McDormand	Fargo
McDormand	Raising Arizona
McDormand	Wonder Boys

Answer

actor
Maguire McDormand

HAVING Clause

Find actors who have acted in more movies than the number of movies made by Hanson

- ❖ An equivalent query without HAVING

```
SELECT  DISTINCT X.actor
FROM    ( SELECT      A.actor AS actor, COUNT (A.title) AS num)
          FROM        Acts A
          GROUP BY  A.actor )  AS X,
        ( SELECT      COUNT (*) AS num
          FROM         Movies M
          WHERE       M.director = 'Hanson' )  AS Y
WHERE   X.num > Y.num
```

HAVING Clause

- Expressions in HAVING clause must have a single value per group
- Each column appearing in HAVING clause must either appear in GROUP BY clause or be an argument of an aggregation operator

Find the number of actors who acted in each movie made after 1998

```
SELECT     M.title, COUNT (A.actor) AS num
FROM       Movies M, Acts A
WHERE      M.title = A.title
GROUP BY M.title
HAVING     M.myear > 1998
```

This query is illegal !

```
SELECT     M.title, COUNT (A.actor) AS num
FROM       Movies M, Acts A
WHERE      M.title = A.title
AND        M.myear > 1998
GROUP BY M.title
```

This query is correct

Ordering the Output

```
SELECT      *
FROM        Movies
ORDER BY  rating DESC, myear ASC
```

Result

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Wonder Boys	Hanson	2000	7.6
Spiderman	Raimi	2002	7.4

Views

- ❖ A view is a table whose rows are not explicitly stored in database
- ❖ A view is a query with a name
- ❖ A view can be used exactly as a table
- ❖ Contents of the view are computed on-the-fly

CREATE VIEW name [schema]
AS sql_query

Views

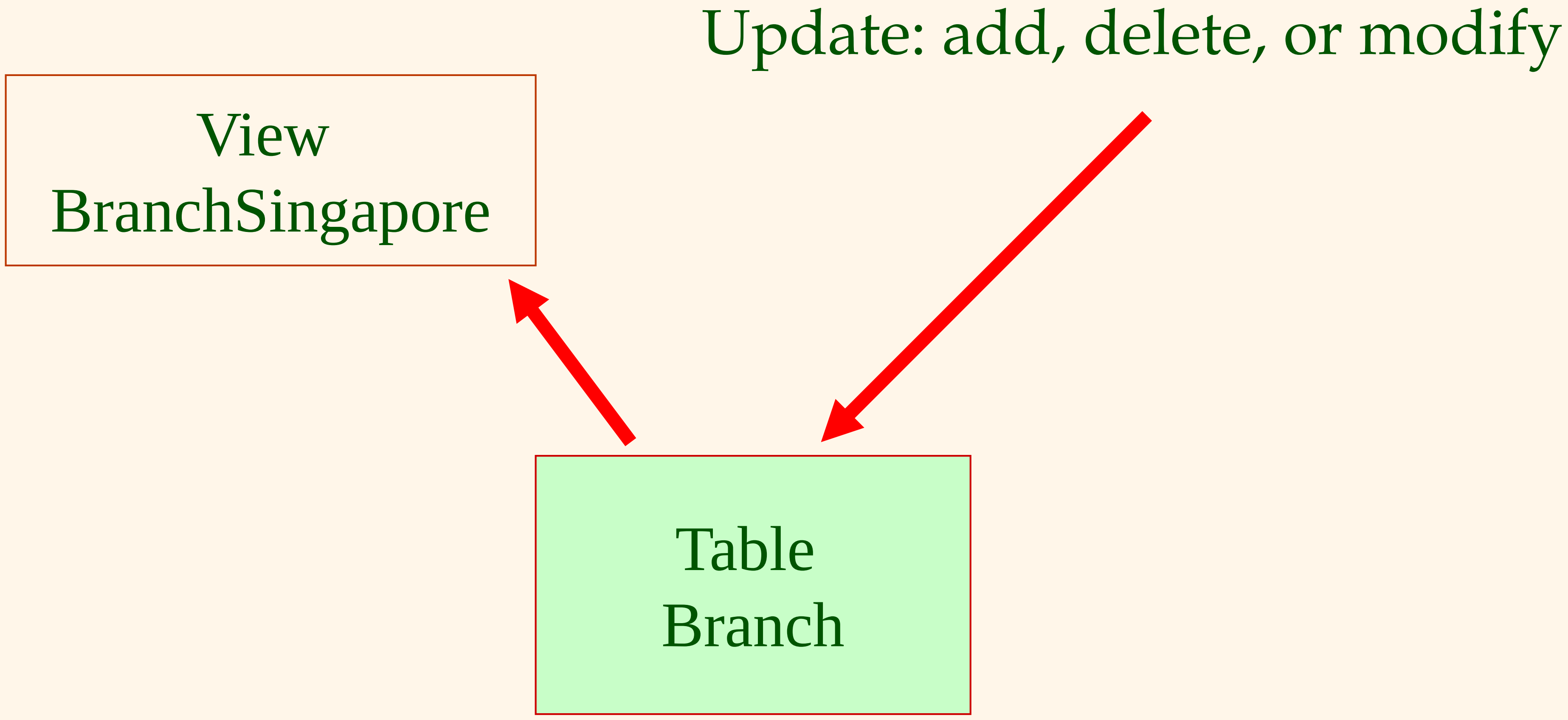
```
CREATE VIEW BranchSingapore
AS
        SELECT  *
        FROM   Branch
        WHERE city = 'Singapore'


SELECT   *   FROM  BranchSingapore
```

Views

name	city	director	assets
Clementi	Kuala Lumpur	Ahmed Abdalah	750000
Clementi	Singapore	Ng Wee Hyong	3000000
East Coast	Singapore	Sanjay Bala	1250000
Jaya	Kuala Lumpur	Putri Bte Alif	9500000
Lion	Singapore	Kevin Hsu	2500000
Monas	Jakarta	Agus Arianto	900000
Twin Towers	Kuala Lumpur	Alif Mohamed	2000000
Wijaya	Jakarta	Oliver Ooi	1200000

name	city	director	assets
Clementi	Singapore	Ng Wee Hyong	3000000
Lion	Singapore	Kevin Hsu	2500000
East Coast	Singapore	Sanjay Bala	1250000

Views


Update: add, delete, or modify
View
BranchSingapore
Table
Branch

Views

name	city	director	assets
Clementi	Kuala Lumpur	Ahmed Abdalah	750000
Clementi	Singapore	Ng Wee Hyong	3000000
East Coast	Singapore	Sanjay Bala	1000000
Jaya	Kuala Lumpur	Putri Bte Alif	9500000
Lion	Singapore	Kevin Hsu	2500000
Monas	Jakarta	Agus Arianto	900000
Twin Towers	Kuala Lumpur	Alif Mohamed	2000000
Wijaya	Jakarta	Oliver Ooi	1200000

name	city	director	assets
Clementi	Singapore	Ng Wee Hyong	3000000
Lion	Singapore	Kevin Hsu	2500000
East Coast	Singapore	Sanjay Bala	1000000

Summary

- SQL is the standard query language for relational DBMS
- Basic form of querying consists of SELECT, FROM and WHERE clauses
- SQL is more expressive than relational algebra
- Views are useful for defining external schemas and support logical data independence