SQL Basic

Prof. Hyuk-Yoon Kwon https://sites.google.com/view/seoultech-bigdata

Contents

- **Today's lecture**
  - Basic single-table queries
  - Multi-table queries

Most parts are based on slides used in Stanford (http://web.stanford.edu/class/cs145)

2. Single-table queries

What you will learn about in this section

1. The SFW query

- Basic form :
  (there are many more bells and whistles)

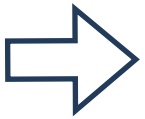
```
SELECT <attributes>
FROM   <one or more relations>
WHERE <conditions>
```

Call this a **SFW** query.

2. Other useful operators: LIKE, DISTINCT, ORDER BY

Simple SQL Query: Selection & Projection

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi



PName	Price	Manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks

Projection

: the operation of producing an output table with tuples that have a subset of their prior attributes

Projection : 특정 Column 추출 = 선택하여 결과 반환

=> 불필요한 data 제거 & 필요한 data만 선택하여 처리 가능

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Selection

: the operation of filtering a relation's tuples on some condition

Selection : 특정 조건 충족하는 열 (row) 선택 = Filtering 작업

=> 필요한 부분만 Data 추출하여 불필요한 데이터 전송 & 처리 방지

```
SELECT  Pname, Price, Manufacturer
FROM    Product
WHERE   Category = 'Gadgets'
```

```
SELECT  *
FROM    Product
WHERE   Category = 'Gadgets'
```

Notation

1) Keyword in Capital ex) SELECT, FROM, WHERE …
2) String with "" & Num without ""

Input schema: Product(PName, Price, Category, Manfacturer)

Output schema: Answer(PName, Price, Manfacturer)

```
SELECT  Pname, Price, Manufacturer
FROM    Product
WHERE   Category = 'Gadgets'
```

Database Schema

```
CREATE TABLE "ACDB_SECTORS"
  (         "SECTOR_ID" NUMBER(8,0),
            "SECTOR_NAME" VARCHAR2(25 BYTE)
  ) ;
```

```
CREATE TABLE "ACDB_PACKAGES"
  (         "PACK_ID" NUMBER(8,0),
            "SPEED" VARCHAR2(10 BYTE),
            "MONTHLY_PAYMENT" NUMBER(8,0),
            "SECTOR_ID" NUMBER(8,0),  "STRT_DATE" DATE
) ;
```

```
CREATE TABLE "ACDB_CUSTOMERS"
  (         "CUSTOMER_ID" NUMBER(8,0),
            "FIRST_NAME" VARCHAR2(25 BYTE),
            "LAST_NAME" VARCHAR2(25 BYTE),
            "CITY"  VARCHAR2(45  BYTE),
            "STATE" VARCHAR2(25 BYTE),
            "STREET" VARCHAR2(40 BYTE),
            "MAIN_PHONE_NUM" VARCHAR2(12 BYTE),
            "SECONDARY_PHONE_NUM" VARCHAR2(12 BYTE),
            "FAX" VARCHAR2(12  BYTE),
            "MONTHLY_DISCOUNT" NUMBER(4,2),
            "PACK_ID" NUMBER(8,0),
            "BIRTH_DATE" DATE,
            "JOIN_DATE" DATE
  ) ;
```

Useful Expressions

- **Arithmetic operation in SELECT clause**
  - SELECT a * 2 FROM table

a
20

- **Alias in SELECT clause**
  - SELECT a as A_RESULT FROM table

A_RESULT
10

a	b
10	20

- **Concatenate in SELECT clause**
  - SELECT a || ‘,’ || b as A_AND_B FROM table

A_AND_B
10, 20

A Few Details

- **SQL commands are case insensitive:**
  - Same: SELECT, Select, select
  - Same: Product, product
- **Values are not:**
  - Different: ‘Seattle’, ‘seattle’
- **Use single quotes for constants:**
  - ‘abc’ - yes
  - “abc” - no

1) LIKE: Simple String Pattern Matching

- **s LIKE p: pattern matching on strings**
- **p may contain two special symbols:**
  - % = any sequence of characters
  - _ = any single character

LIKE 'abc%' : abc로 시작하는 모든 행 반환
LIKE '%abc%' : abc 포함하는 모든 행 반환
LIKE 'a_ _' : a 시작 & 두 글자 임의의 문자열이 뒤에 오는 모든 행 반환

```
SELECT *
FROM   Products
WHERE  PName LIKE '%gizmo%'
```

2) DISTINCT: Eliminating Duplicates

```
SELECT DISTINCT Category F
ROM   Product
```

Category
Gadgets
Photography
Household

VS

```
SELECT Category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

3) ORDER BY: Sorting the Results

```
SELECT  PName, Price, Manufacturer
FROM    Product
WHERE   Category='gizmo'AND Price > 50
ORDER BY  Price, PName
```

Ties are broken by the second attributeon the ORDER BY list, etc.

Ordering is ascending,unless you specify the DESC keyword.

4) BETWEEN operation in WHERE clause

- SELECT a FROM table WHERE a BETWEEN 10 AND 30

a
20
40

a
20

3. Multi-table queries

- **Foreign key constraints**
- **Joins: basics**
- **Joins: SQL semantics**

Primary Key , 기본키

: DB Table에서 특정 Row을 고유하게 식별할 수 있는 column / column들의 집합

-> Table 내 각 Row의 고유성 보장

-> 기본 key 통해 특정 row에 빠르게 access 가능

-> Unique Value (NULL X)

-> Table 내에서 유일 & 일관적 Data 무결성 (Consistent Data Integrity) 유지 가능한 중요한 요소

Foreign Key constraints

- **Suppose we have the following schema:**

  Students(sid: *string*, name: *string*, gpa: *float*)

  Enrolled(student_id: *string*, cid: *string*, grade: *string*)

- **And we want to impose the following constraint:**
  - "a student must appear in the Students table to enroll in a class"

Students

sid	name	gpa
101	Bob	3.2
123	Mary	3.8

Enrolled

student_id	cid	grade
123	564	A
123	537	A+

student_id alone is not a key- what is?

We say that student_id is a **foreign key** that refers to Students

Declaring Foreign Key

Students(sid: *string*, name: *string*, gpa: *float*)
Enrolled(student_id: *string*, cid: *string*, grade: *string*)

```
CREATE TABLE Enrolled(
        student_id CHAR(20),
        cid        CHAR(20),
        grade  CHAR(10),
        PRIMARY KEY (student_id, cid),
        FOREIGN KEY (student_id) REFERENCES Students(sid)
)
```

Foreign Key: used to manage related data across tables (> 1) by creating reference between them

-> consists of column that references primary key column in another table

=> Maintaining Referential Integrity

=> Ensuring data Integrity & Consistency

=> Enhancing DB Safety & Reliability

외래키 : 2개 이상 Table 연결 & 관련 Data 관리하기 위한 기능

-> 다른 Table에서 참조되는 기본 key 열을 참조하는 열로 구성

-> 참조 무결성 유지 위한 중요한 역할

=> Data 무결성 & 일관성 보장 + DB 안전성 & 신뢰성 향상

Foreign Keys and update operations

Students(sid: *string*, name: *string*, gpa: *float*)

Enrolled(student_id: *string*, cid: *string*, grade: *string*)

- **What if we insert a tuple into Enrolled, but no corresponding student?**
  - INSERT is rejected (foreign keys are constraints)!
- **What if we delete a student?**
  1. Disallow the delete

     : 다른 Table에서 외래 키로 참조되는 학생 Record 존재 시 해당 Record 삭제 불가
     = 해당 학생에게 수강 과목이 있다면 삭제 X
  2. Remove all of the courses for that student

     : 해당 학생 Record와 관련된 모든 과목 Record 삭제
     = 해당 학생이 수강한 모든 과목과의 관계 끊을 수 있음
  3. *SQL allows a third via NULL (not yet covered)*

     : 해당 학생 Record와 관련된 모든 과목 Record의 외래 키를 NULL로 설정
     -> 과목 Table에서 Orphaned Records 발생 가능 (주의 필요 - Data 일관성 & 오류 발생 위험)

What is a foreign key vs. a key here?

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Joins

Join : 2개 이상 Table에서 Data 검색 / 결합 시 사용
-> 2개 이상 Table에서 가져온 Data 결합하여 단일 결과 집합으로 반환 가능
-> 특정 Join 조건을 만족하는 Tuples의 모든 고유한 조합 반환
= Join 시 일치하는 값 가지는 Join 조건 충족하는 모든 경우의 수 계산하여 반환
=> 반환되는 Tuples = 각각 두 table에 대한 특정 Row들의 조합

Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)

Ex: Find all products under $200 manufactured in Japan;
return their names and prices.

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
       AND Country='Japan'
       AND Price <= 200
```

Product

PName	Price	Category	Manuf
Gizmo	$19	Gadgets	GWorks
Powergizmo	$29	Gadgets	GWorks
SingleTouch	$149	Photography	Canon
MultiTouch	$203	Household	Hitachi

Company

Cname	Stock	Country
GWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
       AND Country='Japan'
       AND Price <= 200
```

PName	Price
SingleTouch	$149.99

A **join** between tables returns all unique combinations of their tuples
which meet some specified join condition

Several equivalent ways to write a basic join in SQL:

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer = CName
         AND Country='Japan'
         AND Price <= 200
```

```
SELECT   PName, Price
FROM     Product
JOIN     Company ON Manufacturer = Cname
                AND Country='Japan'
WHERE    Price <= 200
```

A few more later on…

Primary Key and Foreign Key

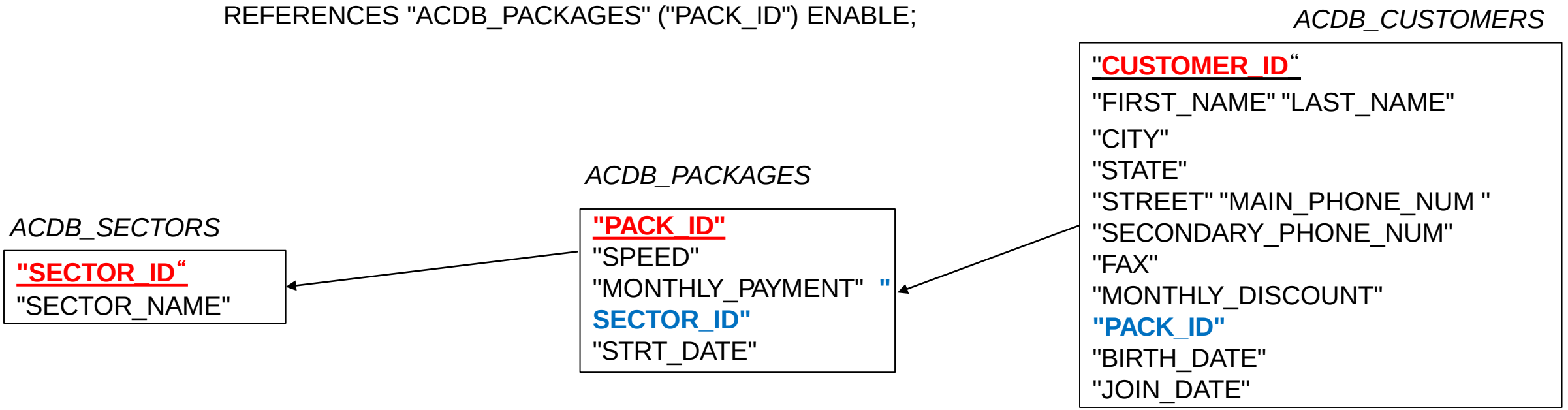
ALTER TABLE "**ACDB_SECTORS**" ADD CONSTRAINT "SECTOR_ID_PK" PRIMARY KEY ("**SECTOR_ID**");

ALTER TABLE "**ACDB_PACKAGES**" ADD CONSTRAINT "PACK_ID_PK" PRIMARY KEY ("**PACK_ID**");

ALTER TABLE "**ACDB_PACKAGES**" ADD CONSTRAINT "SECTOR_ID_FK" FOREIGN KEY ("**SECTOR_ID**")
REFERENCES "ACDB_SECTORS" ("SECTOR_ID") ENABLE;

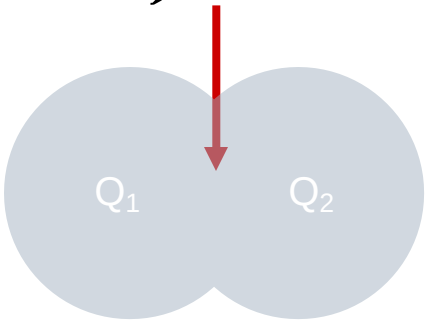
ALTER TABLE "**ACDB_CUSTOMERS**" ADD CONSTRAINT "CUSTOMER_ID_PK" PRIMARY KEY ("**CUSTOMER_ID**");

ALTER TABLE "**ACDB_CUSTOMERS**" ADD CONSTRAINT "PACK_ID_FK" FOREIGN KEY ("**PACK_ID**")
REFERENCES "ACDB_PACKAGES" ("PACK_ID") ENABLE;



Explicit Set Operators: INTERSECT

$$\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}$$



Constraint for Intersect

```
SELECT Name, BirthDate FROM Employee
INTERSECT
SELECT Name, BirthDate FROM Customer
```

```
SELECT Name, BirthDate FROM Employee
INTERSECT
SELECT Age, BirthDate, Name FROM Customer
```

1. 두 SELECT문에서 반환되는 열(column) 수 동일
2. 두 SELECT문에서 반환되는 Data type 동일
3. 중복 행 제거
4. NULL 값 존재 시 제외

```
SELECT   R.A
FROM     R, S
WHERE    R.A=S.A
INTERSECT
SELECT   R.A
FROM     R, T
WHERE    R.A=T.A
```

MINUS

```
SELECT   R.A
FROM     R, S
WHERE    R.A=S.A
   MINUS
SELECT   R.A
FROM     R, T
WHERE    R.A=T.A
```

I have 2 tables A and B.

```
SELECT COUNT(*) FROM (SELECT * FROM tableA)
```

returns 389

```
SELECT COUNT(*) FROM (SELECT * FROM tableB)
```

returns 217

```
SELECT COUNT(*) FROM
(SELECT * FROM tableA
INTERSECT
SELECT * FROM tableB)
```

returns 0

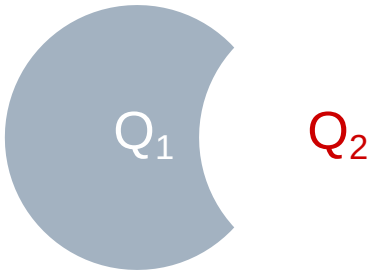
```
SELECT COUNT(*) FROM
(SELECT * FROM tableA
MINUS
SELECT * FROM tableB)
```

returns 389

```
SELECT COUNT(*) FROM
(SELECT * FROM tableB
MINUS
SELECT * FROM tableA)
```

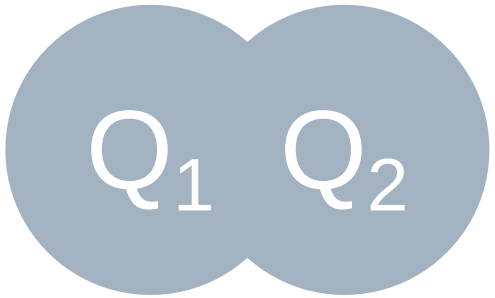
returns 89

$$\{r.A \mid r.A = s.A\} \backslash \{r.A \mid r.A = t.A\}$$



UNION

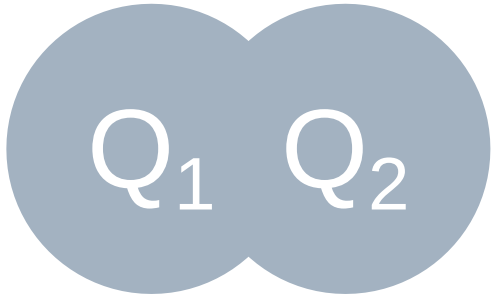
$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$




```
SELECT   R.A
FROM     R, S
WHERE    R.A=S.A
UNION
SELECT   R.A
FROM     R, T
WHERE    R.A=T.A
```

UNION ALL

$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$




```
SELECT   R.A
FROM     R, S
WHERE    R.A=S.A
UNION ALL
SELECT   R.A
FROM     R, T
WHERE    R.A=T.A
```

By default: SQL uses set semantics

Why aren't there duplicates?

UNION removes duplicate rows

What if we want duplicates?

use "UNION ALL"

1. Employee table data:

```
SELECT * FROM Employee
```

Results | Messages

	EmpId	EmpName	EmpCode
1	1	Bhaumik	1609
2	2	Maulik	2568
3	3	Mayur	1254
4	4	Sandip	6578
5	5	Rekansh	7998

2. Customer table data:

```
SELECT * FROM Customer
```

Results | Messages

	CustId	CustName	CustCode
1	1	Bhoms	234
2	2	Mayur	656
3	3	Jimit	324
4	4	Sandip	435

3. UNION Example (It removes all duplicate records):

```
SELECT EmpName FROM Employee
UNION
SELECT CustName FROM Customer
```

Results | Messages

	EmpName
1	Bhaumik
2	Bhoms
3	Jimit
4	Maulik
5	Mayur
6	Rekansh
7	Sandip

4. UNION ALL Example (It just concatenate records, not eliminate duplicates, so it is faster than UNION):

```
SELECT EmpName FROM Employee
UNION ALL
SELECT CustName FROM Customer
```

Results | Messages

	EmpName
1	Bhaumik
2	Maulik
3	Mayur
4	Sandip
5	Rekansh
6	Bhoms
7	Mayur
8	Jimit
9	Sandip