MORE SQL

Introduction to Database Systems

Mohammad Tanhaei
Ilam University

INSERT, UPDATE, DELETE

- **INSERT** - add a row to a table
- **UPDATE** - change row(s) in a table
- **DELETE** - remove row(s) from a table

- UPDATE and DELETE use 'WHERE clauses' to specify which rows to change or remove
- BE CAREFUL with these - an incorrect **`WHERE`** clause can destroy lots of data

INSERT

```
INSERT INTO
  <table>
  (col1, col2, …)
  VALUES
  (val1, val2, …)
```

- The number of columns and values must be **the same**
- If you are adding a value to every column, you don't have to list them
- SQL doesn't require that all rows are different (unless a constraint says so)

INSERT

```
INSERT INTO Student
 (ID, Name, Year)
 VALUES (2, 'Mary', 3)
```

Student

ID	Name	Year
1	John	1
2	Mary	3

Student

ID	Name	Year
1	John	1

```
INSERT INTO Student
 (Name, ID)
 VALUES ('Mary', 2)
```

Student

ID	Name	Year
1	John	1
2	Mary	1

```
INSERT INTO Student
 VALUES (2, 'Mary', 3)
```

Student

ID	Name	Year
1	John	1
2	Mary	3

UPDATE

```
UPDATE <table>
SET col1 = val1
  [,col2 = val2…]
[WHERE
  <condition>]
```

- All rows where the condition is true have the columns set to the given values
- If no condition is given all rows are changed so **BE CAREFUL**
- Values are constants or can be computed from columns

UPDATE

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Mary	2

```
UPDATE Student
SET Year = 1,
    Name = 'Jane'
WHERE ID = 4
```

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Jane	1

```
UPDATE Student
SET Year = Year + 1
```

Student

ID	Name	Year
1	John	2
2	Mark	4
3	Anne	3
4	Mary	3

DELETE

- Removes all rows which satisfy the condition

```
DELETE FROM
   <table>
  [WHERE
   <condition>]
```

- If no condition is given then ALL rows are deleted - **BE CAREFUL**
- Some versions of SQL also have `TRUNCATE TABLE <T>` which is like `DELETE FROM <T>` but it is quicker as it doesn't record its actions

DELETE

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Mary	2

```
DELETE FROM
  Student
  WHERE Year = 2
```

Student

ID	Name	Year
1	John	1
2	Mark	3

```
DELETE FROM Student
        or
TRUNCATE TABLE Student
```

Student

ID	Name	Year

SELECT

- The SQL command you will use most often
  - Queries a set of tables and returns results as a table
  - Lots of options, we will look at many of them
  - Usually more than one way to do any given query
- SQL's SELECT is different from the relational algebra's selection σ
  - SELECT in SQL does all of the relational algebra
  - But it is a bit different because SQL differs from the relational model

SQL SELECT OVERVIEW

```
SELECT
   [DISTINCT | ALL] <column-list>
   FROM <table-names>
   [WHERE <condition>]
   [ORDER BY <column-list>]
   [GROUP BY <column-list>]
   [HAVING <condition>]
```

- *([]- optional, | - or)*

SIMPLE SELECT

```
SELECT <columns>
   FROM <table>
```

`<columns>` can be

- A single column
- A comma-separated list of columns
- * for ‘all columns’

- Given a table Student with columns
  - stuID
  - stuName
  - stuAddress
  - stuYear

SAMPLE SELECTS

- SELECT * FROM Student

stuID	*stuName*	*stuAddress*	*stuYear*
1	Anderson	15 High St	1
2	Brooks	27 Queen's Rd	3
3	Chen	Lenton Hall	1
4	D'Angelo	Derby Hall	1
5	Evans	Lenton Hall	2
6	Franklin	13 Elm St	3
7	Gandhi	Lenton Hall	1
8	Harrison	Derby Hall	1

SAMPLE SELECTS

```
SELECT stuName FROM Student
```

stuName
Anderson
Brooks
Chen
D'Angelo
Evans
Franklin
Gandhi
Harrison

SAMPLE SELECTS

```
SELECT stuName, stuAddress
   FROM Student
```

stuName	*stuAddress*
Anderson	15 High St
Brooks	27 Queen's Rd
Chen	Lenton Hall
D'Angelo	Derby Hall
Evans	Lenton Hall
Franklin	13 Elm St
Gandhi	Lenton Hall
Harrison	Derby Hall

BEING CAREFUL

- When using DELETE and UPDATE
  - You need to be careful to have the right WHERE clause
- You can check it by running a SELECT statement with the same WHERE clause first
- Before running

```
DELETE FROM Student
    WHERE Year = 3
```

- Run

```
SELECT * FROM Student
    WHERE Year = 3
```

SEQUENCES

- Often we want to assign each row a unique number
  - These are useful as primary keys
  - Using integers to reference rows is more efficient
  - We would like the DBMS to do this
- In most versions of SQL we can use **autoincrementing** fields to do this
  - Details differ between versions
  - Usually the first entry is assigned 1, the next 2, and so on, but Oracle lets you change this

SEQUENCES

- In Oracle we use a *Sequence*
  - A sequence is a source of numbers
  - We can declare several sequences, giving each a name, a start point, and a step size
  - We can then generate unique numbers by asking for the next element from a sequence

SEQUENCES IN ORACLE

- To declare a sequence:

```
CREATE SEQUENCE <name>
    [START WITH <value>]
    [INCREMENT BY <value>]
```

  - If no **START WITH** or **INCREMENT BY** values are given they default to 1
- To get the next value from a sequence

```
<sequence name>.nextVal
```

SEQUENCE EXAMPLE

- Creating a sequence

```
CREATE SEQUENCE mySeq START WITH 1
```

- Using a sequence

```
SELECT mySeq.nextVal FROM DUAL;
```

*The **DUAL** table is a special **one-row, one-column** table present by default in Oracle and other database installations.*

```
INSERT INTO Student
   (stuID, stuName, stuAddress)
VALUES
   (mySeq.nextVal, 'Steve Mills',
    '13 Elm Street')
```

SQL AND THE DATA DICTIONARY

- The *data dictionary* or *catalogue* stores
  - Information about database tables
  - Information about the columns of tables
  - Other information - users, locks, indexes, and more
- This is 'metadata'
- Some DBMSs let you query the catalogue
  - In Oracle you can access the metadata in several ways
    - There are 'system tables' with metadata in them
    - You can also **`DESCRIBE`** tables

ORACLE DATA DICTIONARY

- To find out what tables and sequences you have defined use

```
SELECT table_name
   FROM user_tables
```

- The user_tables table is maintained by Oracle
- It has *lots* of columns, so don't use

```
SELECT * FROM user_tables
```

ORACLE DATA DICTIONARY

- To find the details of a table use

```
DESCRIBE <table name>
```

- Example:

```
SQL> DESCRIBE Student;
Name            Null?    Type
--------------- -------- ----------
STUID           NOT NULL NUMBER(38)
STUNAME         NOT NULL VARCHAR2(50)
STUADDRESS               VARCHAR2(50)
STUYEAR                  NUMBER(38)
```

SQL SELECT OVERVIEW

```
SELECT
   [DISTINCT | ALL] <column-list>
  FROM <table-names>
   [WHERE <condition>]
   [ORDER BY <column-list>]
   [GROUP BY <column-list>]
   [HAVING <condition>]
```

- *([]- optional, | - or)*

EXAMPLE TABLES

Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	Jones
S107	John	Brown

Course

Code	Title
DBS	Database Systems
PR1	Programming 1
PR2	Programming 2
IAI	Intro to AI

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

DISTINCT AND ALL

- Sometimes you end up with duplicate entries
- Using **`DISTINCT`** removes duplicates
- Using **`ALL`** retains them - this is the default

```
SELECT ALL Last
  FROM Student
```

Last
Smith
Jones
Brown
Jones
Brown

```
SELECT DISTINCT Last
  FROM Student
```

Last
Smith
Jones
Brown

WHERE CLAUSES

- Usually you don't want all the rows
  - A **`WHERE`** clause restricts the rows that are returned
- It takes the form of a condition - only those rows that satisfy the condition are returned
- Example conditions:
  - Mark < 40
  - First = 'John'
  - First <> 'John'
  - First = Last
  - (First = 'John') AND (Last = 'Smith')
  - (Mark < 40) OR (Mark > 70)

WHERE EXAMPLES

```
SELECT * FROM Grade
 WHERE Mark >= 60
```

ID	Code	Mark
S103	DBS	72
S104	PR1	68
S104	IAI	65
S107	PR1	76
S107	PR2	60

```
SELECT DISTINCT ID
  FROM Grade
  WHERE Mark >= 60
```

ID
S103
S104
S107

WHERE EXAMPLE

- Given the table

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

- Write an SQL query to find a list of the ID numbers and marks in IAI of students who have passed (scored 40 or higher) IAI

ID	Mark
S103	58
S104	65

ONE SOLUTION

We only want the ID and Mark, not the Code

Single quotes around the string

```
SELECT ID, Mark FROM Grade
  WHERE (Code = 'IAI') AND
  (Mark >= 40)
```

We're only interested in IAI

We're looking for entries with pass marks

SELECT FROM MULTIPLE TABLES

- Often you need to combine information from two or more tables
- You can get the effect of a product by using

```
SELECT * FROM Table1,
  Table2...
```

- If the tables have columns with the same name ambiguity results
- You resolve this by referencing columns with the table name

```
TableName.Column
```

SELECT FROM MULTIPLE TABLES

```
SELECT
   First, Last, Mark
FROM Student, Grade
WHERE
   (Student.ID =
      Grade.ID) AND
   (Mark >= 40)
```

Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	
S107	John	

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

SELECT FROM MULTIPLE TABLES

```
SELECT ... FROM Student, Grade WHERE...
```

Are matched with the first entry from the Student table...

And then with the second…

and so on

ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S103	John	Smith	S104	PR1	68
S103	John	Smith	S104	IAI	65
S103	John	Smith	S106	PR2	43
S103	John	Smith	S107	PR1	76
S103	John	Smith	S107	PR2	60
S103	John	Smith	S107	IAI	35
S104	Mary	Jones	S103	DBS	72
S104	Mary	Jones	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65
S104	Mary	Jones	S106	PR2	43

All of the entries from the Grade table

SELECT FROM MULTIPLE TABLES

```
SELECT ... FROM Student, Grade
  WHERE (Student.ID = Grade.ID) AND ...
```

ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65
S106	Mark	Jones	S106	PR2	43
S107	John	Brown	S107	PR1	76
S107	John	Brown	S107	PR2	60
S107	John	Brown	S107	IAI	35

Student.ID

Grade.ID

SELECT FROM MULTIPLE TABLES

```
SELECT ... FROM Student, Grade
  WHERE (Student.ID = Grade.ID) AND (Mark >= 40)
```

ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65
S106	Mark	Jones	S106	PR2	43
S107	John	Brown	S107	PR1	76
S107	John	Brown	S107	PR2	60

SELECT FROM MULTIPLE TABLES

```
SELECT First, Last, Mark FROM Student, Grade
  WHERE (Student.ID = Grade.ID) AND (Mark >= 40)
```

First	Last	Mark
John	Smith	72
John	Smith	58
Mary	Jones	68
Mary	Jones	65
Mark	Jones	43
John	Brown	76
John	Brown	60

SELECT FROM MULTIPLE TABLES

- When selecting from multiple tables you almost always use a **`WHERE`** clause to find entries with common values

```
SELECT * FROM
  Student, Grade, Course
WHERE
  Student.ID = Grade.ID
AND
  Course.Code = Grade.Code
```

SELECT FROM MULTIPLE TABLES

Student | Grade | Course

ID	First	Last	ID	Code	Mark	Code	Title
S103	John	Smith	S103	DBS	72	DBS	Database Systems
S103	John	Smith	S103	IAI	58	IAI	Intro to AI
S104	Mary	Jones	S104	PR1	68	PR1	Programming 1
S104	Mary	Jones	S104	IAI	65	IAI	Intro to AI
S106	Mark	Jones	S106	PR2	43	PR2	Programming 2
S107	John	Brown	S107	PR1	76	PR1	Programming 1
S107	John	Brown	S107	PR2	60	PR2	Programming 2
S107	John	Brown	S107	IAI	35	IAI	Intro to AI

Student.ID = Grade.ID

Course.Code = Grade.Code

JOINS

- JOINs can be used to combine tables
  - There are many types of JOIN
    - **CROSS JOIN**
    - **INNER JOIN**
    - **NATURAL JOIN**
    - **OUTER JOIN**
  - OUTER JOINs are linked with NULLs - more later
- A CROSS JOIN B
  - returns all pairs of rows from A and B
- A NATURAL JOIN B
  - returns pairs of rows with common values for identically named columns and **without duplicating columns**
- A INNER JOIN B
  - returns pairs of rows satisfying the condition

CROSS JOIN

```
SELECT * FROM
   Student CROSS JOIN
   Enrolment
```

Enrolment

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

ID	Name	ID	Code
123	John	123	DBS
124	Mary	123	DBS
125	Mark	123	DBS
126	Jane	123	DBS
123	John	124	PRG
124	Mary	124	PRG
125	Mark	124	PRG
126	Jane	124	PRG
123	John	124	DBS
124	Mary	124	DBS

NATURAL JOIN

```
SELECT * FROM
   Student NATURAL JOIN Enrolment
```

Enrolment

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

ID	Name	Code
123	John	DBS
124	Mary	PRG
124	Mary	DBS
126	Jane	PRG

CROSS AND NATURAL JOIN

```
SELECT * FROM
  A CROSS JOIN B
```

is the same as

```
SELECT * FROM A, B
```

```
SELECT * FROM
  A NATURAL JOIN B
```

is the same as

```
SELECT A.col1,… A.coln,
[and all other columns
apart from B.col1,…B.coln]
FROM A, B
WHERE A.col1 = B.col1
  AND A.col2 = B.col2
...AND A.coln = B.col.n
(this assumes that col1…
coln in A and B have
common names)
```

INNER JOIN

INNER JOINs specify a condition which the pairs of rows satisfy

```
SELECT * FROM
  A INNER JOIN B
  ON <condition>
```

- Can also use

```
SELECT * FROM
  A INNER JOIN B
  USING
    (col1, col2,…)
```

- Chooses rows where the given columns are equal

INNER JOIN

```
SELECT * FROM
   Student INNER JOIN Enrolment USING (ID)
```

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG

```
SELECT * FROM
   Student INNER JOIN
   Enrolment USING (ID)
```

ID	Name	ID	Code
123	John	123	DBS
124	Mary	124	PRG
124	Mary	124	DBS
126	Jane	126	PRG

INNER JOIN

```
SELECT * FROM
   Buyer INNER JOIN Property ON
   Price <= Budget
```

Buyer

Name	Budget
Smith	100,000
Jones	150,000
Green	80,000

Property

Address	Price
15 High St	85,000
12 Queen St	125,000
87 Oak Row	175,000

```
SELECT * FROM
Buyer INNER JOIN
 Property ON
Price <= Budget
```

Name	Budget	Address	Price
Smith	100,000	15 High St	85,000
Jones	150,000	15 High St	85,000
Jones	150,000	12 Queen St	125,000

INNER JOIN

```
SELECT * FROM
   A INNER JOIN B
   ON <condition>
```

- is the same as

```
SELECT * FROM A, B
   WHERE <condition>
```

```
SELECT * FROM
    A INNER JOIN B
  USING(col1,
col2,...)
```

- is the same as

```
SELECT * FROM A, B
  WHERE A.col1 =
B.col1
    AND A.col2 =
B.col2
    AND ...
```

JOINS VS WHERE CLAUSES

- JOINs (so far) are not needed
  - You can have the same effect by selecting from multiple tables with an appropriate WHERE clause
- So should you use JOINs or not?
  - Yes, because
    - They often lead to **concise** queries
    - NATURAL JOINs are very common
  - No, because
    - Support for JOINs varies a fair bit among SQL dialects

WRITING QUERIES

- When writing queries
  - There are often many ways to write the query
  - You should worry about being correct, clear, and concise in that order
  - Don't so much worry about being clever or efficient
- Most DBMSs have query optimizers
  - These take a user's query and figure out how to efficiently execute it
  - A simple query is easier to optimize
  - We'll look at some ways to improve efficiency later

THIS LECTURE IN EXAMS

Track

cID	Num	Title	Time	aID
1.	1	Violent	239	1
1	2	Every Girl	410	1
1.	3	Breather	217	1
1	4	Part of Me	279	1
2.	1	Star	362	1
2	2	Teaboy	417	2

CD

cID	Title	Price
1.	Mix	9.99
2.	Compilation	12.99

Artist

aID	Name
1.	Stellar
2.	Cloudboy

THIS LECTURE IN EXAMS

- Add £2.50 to the price of all CDs that cost more than £10.00. (20 marks)
- Add a new column, Genre, to the CD table. This column should hold a string of up to 100 characters, and if no genre is provided then it should default to the value "Unknown". (30 marks)
- Add a track titled "Runnin" by the artist "Fat Freddy's Drop" which is 12 minutes and 27 second long to the CD titled "Compilation". For this part only, you may assume that the tables contain exactly the information shown above. (30 marks)

THIS LECTURE IN EXAMS

- Find a list of all the CD titles. (10 mark)
- Find a list of the titles of tracks that are more than 300 seconds long. (20 marks)
- Find a list of the names of those artists who have a track on the CD with the title “Compilation”. (40 marks)

END

But take a look at next slide!

NEXT LECTURE

- Event More SQL SELECT!
  - Aliases
  - 'Self-joins'
  - Subqueries
    - IN, EXISTS, ANY, ALL
- For more information
  - Connolly and Begg Chapter 5
  - Ullman and Widom Chapter 6