Chapter 4

Intermediate SQL

Join Condition:

Natural Join require values to match specified attributes.

SQL supports another join in which an attribute join condition can be specified.

The on condition allows a general predicate over the relations being joined. This predicate is written like a where clause predicate except for the use of the keyword "on" rather than "where".

"On" condition appears at the end of join expression.

"Find students and courses taken in university.

```
Select * from student join takes on
  student.ID = takes.ID.
```

Using where clause, the above query is equal to following:

```
Select *
from student, takes
where student.ID = takes.ID.
```

The "on" condition can express any SQL predicate, and thus a join expressions using the "on" condition can express richer class of join conditions than natural join.

It appears that on condition is a redundant feature of SQL.

Two reasons to use "on" condition:

- A kind of join (see later) called outer join use "on" which behave different than where clause.
- SQL is more readable by human if join condition is specified in on clause and the rest of the condition appears in the where clause.

⊞ Outer Join:

Display list of all students, displaying their ID, and name, dept-name & total credit along with the courses that they have taken.

```
select * from student natural join
takes;
```

The above query is not sufficient to express

Suppose, there are some students who take no courses. Then tuple of student relation for that particular student would not satisfy the condition of a natural join with any tuple in the take relation and that student's data would not appear in the result. Thus we would not see any info. about students who have not taken any courses.

The outer join works in a manner similar to the join operation studied earlier, but preserve those tuples that would be lost in a join, by creating tuples in the result containing null values.

In student relation, ~~so~~ 'snow' doesn't take any courses and thus natural join omit this tuple in result. But using 'outer join' the result shows snow tuple but all attributes in teaches relation assigns "null".

Three forms of outer join:

- Left outer join: preserves tuples only in the relation named before (left) the left outer join.

- The right outer join preserves tuples only in the relation named after (right) the right outer join.
- The full outer join preserves tuples in both relation.

In contrast, the join operation don't preserve non matched tuples are called inner join operations.

☒ Left outer join: At first, it computes the result of the inner join as before. Then for every tuple 't' in left hand side relation that doesn't match any tuple in the right hand side relation in the inner join, add a tuple r to the result of the join constructed as follows:

- The attributes of tuple r that are derived from the left hand side relation are filled in with the values from tuple t.
- The remaining attributes of r are filled with null values.

```
select *
from student natural left outer join takes;
```

includes nulls for the attributes that appears only in the schema of the takes relation.

Ex: Find all students who have not taken any courses.

```
Select ID
from student natural left outer join takes
where course_id is null;
```

- Right Outer Join: Symmetric to left outer join. Tuples from the right hand side relation that don't match any tuple in the left hand side relation are padded with nulls and are added to the result of the right outer join.

```
select *
from takes natural right outer join student;
```

- Full outer Join: Combination of left and right outer join types. After the operation computes the result of the inner join, it extends with nulls those tuples from the left hand side relation that didn't match with any from the right those tuples from the right hand side rel^n that didn't match with any tuples from the left hand side & added them to result.

Display a list of all students in the Comp. Sci department, along with the course sections, if any, that they have taken in spring 2009; all course sections from spring 2009 must be displayed, even if no student from Comp. Sci department has taken the course section.

```
select *
~~&~~ from (select *
       from student
       where dept_name = 'Comp. Sci')
      natural full outer join
      (select *
       from takes
       where semester = 'spring' and
       year = 2009);
```

Chapter 4 (2)

Transactions

It consists of a sequence of query and/or update statements. The SQL standard specifies that a transaction begins implicitely when an SQL statement is executed. One of the following SQL statements must end the transaction:

Commit work: Commits the current transaction; that is it makes the update performed by the transaction become permanent in the database. After the transaction is committed, a new transaction is automatically started.

Rollback work: causes the current transaction to be rolled back; that is it undoes all the updates performed by the SQL statements in the transaction. Thus, the database state is restored to what it was before the first statement of transaction was executed.

Integrity Constraints: It ensures that changes made to the database by authorized users donot result in a loss of data consistency. Thus integrity constraints guard against accidental damage to database.

Ex:

- Instructor name can't be null.
- No two instructor can have same id.
- Each department name in the course rel^n must have a matching department name in the department rel^n.
- The budget of a department must be greater than 0.

Constraints on a single relation.

Besides primary key, there are a number of other constraints in single relation. They are:

- not null
- unique
- check (< predicate>)

Not Null: Null values is a member of all domains, and legal values for every attributes in SQL by default. But in some cases it is inappropriate like name & budget. If name allows null, then a tuple of student information, with no name is not suitable. Thus it is appropriate to use not null which can be defined as follows:

name varchar (20) not null.

It prohibits the insertion of a null value for the

attribute. Any attempt to insert null in a not null attribute would cause error.

Unique Constraints: SQL also support integrity constraint : unique (Aj1, Aj2, ... Ajn);

The unique specification says that attributes Aj1, Aj2, ... Ajn form candidate key; that is, no two tuples in the relation can be equal on all the listed attributes. However, candidate key attributes are permitted to be null unless they are explicitely been declared to be not null.

Check Clause: It specifies a predicate P that must be satisfy by every tuple in a relation.

A common use of the check clause is to ensure that attribute value satisfy specified condition, in effect creating a powerful type system.

Ex: check (budget>0) in the creat table command for relation department would ensure that the value of budget is non negetive.

```
creat table department
(
    dept_name   varchar (100),
    building    varchar (50),
    budget      int,
    primary key (dept_name),
```

check(budget>0));

SQL datatype & schemas:

date: A calendar date containing a (four digit) year, month and a day of the month.

time: The time of the day in hours, minutes and seconds. A varient, time(p), can be used to specify the number of fractional digits for seconds. It is also possible to store timezone information along with the time by specifying time with timezone.

timestamp: A combination of date and time. A varient, timestamp(p), can be used to specify the number of fractional ~~part~~ digits for seconds.

Ex:
```
date '2016-03-21'
time '09:30:45'
timestamp '2016-03-21 09:30:45'
```

Default Values: SQL allows a default value to be specified for an attribute as follows;

```
create table student
( id   varchar(5),
  name   varchar (20) not null,
  tot_cred  numeric (3,0) default 0,
  primary key (id));
```

When tuple is inserted into student relation, no tuple is provided for tot_cred attribute, its value is set to zero (0).

```
insert into student (id, name)
values ('12789', 'Nawman');
```

Authorization :

We may assign a user several forms of authorization on parts of the database. It includes:

- Authorization to read data.
- " " insert "
- " to update "
- " " delete "

Each of these types of authorization is called privilages. We may authorize the user all, none, or a combination of these types of privilages on specified parts of a database.

When a user submits a query or an update, the SQL implementation first checks if the query or update ~~to~~ is authorized, based on the authorizations that the user has been granted. If updates is not authorized, then query is rejected.

In addition to authorization, on data, users may also be granted authorization on database schem allowing them, for example, to create, modify or drop relation. A user who has some form of authorization may be allowed to pass on (gran this authorization to other users, or to withdra (revoke) an authorization that was granted earlier.

Granting & revoking of privilages:

SQL data definition language includes command to grant and revoke privilages.

The basic form of grant statement:

```
grant <privilage list>  ← insert, update, delete,
                               update
on < relation name or view name>
to <user/role list>
```

Example:

```
grant select on department to amit,
   satoshi;
```

This allows those users to run queries on the department relation.

To revoke an authorization, we use statement as follows,

```
revoke <privilage list>
on <relation name or view name>
from <user/role list>
```

For example:

```
revoke select on department from
amit, satoshi;
```

```
Begin
Begin Try
Begin Transaction

update ...
update
commit transaction
print 'Tran complete'
End try
Begin catch
Rollback transaction
print 'Roll back'
End catch
end
```