Relational DBMS & SQL

- Data Definition in SQL
- Indexes in SQL
- Database Queries in SQL
- Aggregate Functions of SQL
- Special Features of SQL
- Database Update in SQL

RDBMS, SQL Statements

- Relational DBMS must
  - Allow to define and manipulate relations
  - Manage DB in terms of relations
  - Accept normalized relations

- SQL (structured query language)
  - ANSI and ISO standard for relational DB
  - Supported by other non-relational DB systems

- SQL statements
  - Data definition:
    - create table, alter table, drop table, create view, drop view, create index, drop index
  - Data manipulation:
    - select, insert, delete
A Relation Example

<table>
<thead>
<tr>
<th>Student</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>s1</td>
<td>Jose</td>
</tr>
<tr>
<td>s2</td>
<td>Alice</td>
</tr>
<tr>
<td>s3</td>
<td>Tom</td>
</tr>
<tr>
<td>s4</td>
<td>Sue</td>
</tr>
<tr>
<td>s5</td>
<td>Steve</td>
</tr>
</tbody>
</table>

Create Relations

```
CREATE TABLE TableName
(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [...] )
[PRIMARY KEY (listOfColumns),]
[UNIQUE (listOfColumns),] [...]]]
[FOREIGN KEY (listOfFKColumns)
REFERENCES ParentTableName [(listOfCKColumns)],
[ON UPDATE referentialAction]
[ON DELETE referentialAction] [...] ]
[CHECK (searchCondition)] [...] )
```

BNF Notation:
- All caps or boldface indicate required reserved words
- Mixed case or italics indicate user-supplied values
- Square brackets indicates optional
- Parentheses are significant. Must be included if shown
- Bar character ("|") indicates an or condition

Classrooms

<table>
<thead>
<tr>
<th>EMP#</th>
<th>SID</th>
<th>Course#</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>s1</td>
<td>CIS2010</td>
<td>12:10 pm</td>
<td>100-CS</td>
</tr>
<tr>
<td>c3</td>
<td>s3</td>
<td>CIS3210</td>
<td>8:00 am</td>
<td>521-K</td>
</tr>
<tr>
<td>c2</td>
<td>s2</td>
<td>CIS3215</td>
<td>4:30 pm</td>
<td>327-ALS</td>
</tr>
<tr>
<td>c3</td>
<td>s1</td>
<td>CIS3215</td>
<td>4:30 pm</td>
<td>327-ALS</td>
</tr>
<tr>
<td>c1</td>
<td>s3</td>
<td>CIS3210</td>
<td>8:00 am</td>
<td>521-K</td>
</tr>
<tr>
<td>c4</td>
<td>s4</td>
<td>CIS3730</td>
<td>7:15 pm</td>
<td>621-G</td>
</tr>
<tr>
<td>c2</td>
<td>s2</td>
<td>CIS3000</td>
<td>4:30 pm</td>
<td>327-S</td>
</tr>
<tr>
<td>c1</td>
<td>s2</td>
<td>CIS2010</td>
<td>12:10 pm</td>
<td>100-CS</td>
</tr>
<tr>
<td>c1</td>
<td>s2</td>
<td>CIS2010</td>
<td>12:10 pm</td>
<td>100-CS</td>
</tr>
</tbody>
</table>

CIS
Create Relations (alternate form)

Create Table TableName
(colName dataType [Not Null] [Unique]
[Default defaultOption]
[Check searchCondition] [,...] ]
[Primary Key (listOfColumns),]
[Unique (listOfColumns)] [,...]]
[Foreign Key (listOfFKColumns)
References ParentTable [listOfCKColumns],
[On Update referentialAction]
[On Delete referentialAction]] [,...]]
[Check (searchCondition)] [,...] )

BNF Notation:
• All caps or boldface indicate required reserved words
• Mixed case or italics indicate user-supplied values
• Square brackets indicates optional
• Parentheses are significant. Must be included if shown
• Bar character ("|") indicates an or condition

Create Relations (cont’d)

Create table AUTHORS
(AUID NVARCHAR2(10) NOT NULL,
BOOKNO NVARCHAR2(13) NOT NULL,
AUNAME NVARCHAR2(25),
PRIMARY KEY (AUID),
FOREIGN KEY (BOOKNO)
REFERENCES BOOKS
ON DELETE CASCADE);

Attribute types of Oracle:
integer, number( p, q ), char( n ),
varchar2(n), nchar(n), nvarchar2(n),
date( dd-monthabbrev-yyyy ), long,
raw(p), long raw, rowid.
DB2 types accepted by Oracle:
smallint, decimal(p,q), float,
varchar(n), long, varchar

☐ The NULL value represents a unknown value or is not applicable.

A null value can be used for the GPA for a new student
CREATE TABLE

- Creates a table with one or more columns of the specified data type.
- With NOT NULL, system rejects any attempt to insert a null in the column.
- Can specify a DEFAULT value for the column.
- Primary keys should always be specified as NOT NULL.
- FOREIGN KEY clause specifies FK along with the referential action.

ALTER Relation

ALTER TABLE base-relation-name
ADD attr-name data-type;

DROP TABLE base-relation-name;

alter table STUDENT
  add AptNum char(5);

drop table STUDENT;

- New attribute values in existing tuples are initialized to null values
- Values for the new attribute must be set by the user
- Only one attribute may be added at a time
Indexes

There is no order among tuples
All tuples in a relation may be sorted by the primary key attribute values

Indexes speed up data retrieval
How many student’s tuples would we search for a query with and without indexes?

Find all students who live in Atlanta?

<table>
<thead>
<tr>
<th>AddrIndex</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlanta</td>
<td></td>
</tr>
<tr>
<td>Boston</td>
<td></td>
</tr>
<tr>
<td>New York</td>
<td></td>
</tr>
</tbody>
</table>

Index Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Joseph</td>
<td>Boston</td>
<td>3.7</td>
</tr>
<tr>
<td>s2</td>
<td>Alice</td>
<td>Atlanta</td>
<td>4.0</td>
</tr>
<tr>
<td>s3</td>
<td>Tom</td>
<td>New York</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Create Indexes

CREATE [UNIQUE] INDEX index name ON base-relation-name ( attr-name [order], attr-name[order] ... ) [CLUSTER];

- An index may consist of one or more attributes
  Example: Address/Age index
- The Unique option forces unique index tuples
  The primary key attribute of a relation can be simulated as an indexed attribute with Unique option created by the DBMS.
- The Cluster option groups index in a physical block
- An ascending order is by default
### Drop Indexes

create unique index student-id
   on STUDENT ( ID ASC )
   CLUSTER;

create index Address-index
   on Student (Address)

create unique index Name-Index
   on STUDENT ( Name DESC, Age )
   CLUSTER;

DROP INDEX index-name;

drop index Address-Index;

What are the major advantages and disadvantages of using index?

---

### Data Retrieval

**SELECT [ DISTINCT ] \[ attribute1 [,attr2, attr3,...] \] FROM \[ relation1 [, relation2, relation3,...] \] [ WHERE condition ] [ GROUP BY attribute1 [,attr, ...] ] [ Having condition ] [ order by attribute1 [,attr2,...] [ASC|DESC] ] ;

**Projection:**

List all student names and addresses.

```
select Name, Address 
from STUDENT;
```

**Restriction:**

List all student with GPA > 3.0.

```
select * 
from STUDENT 
where GPA > 3.0;
```
**Data Retrieval...**

**Project & Restriction:**
List all student names and addresses who have GPA > 3.0 and age > 20.

```
select distinct Name, Address
from STUDENT
where GPA > 3.0 and Age > 20
order by Name DESC
```

- SQL does not remove duplicated tuples
- *Distinct* requests to remove duplicate tuples
- *Order by* sorts the result in ascending or descending order

---

**Join**

**Extended Cartesian Product:**
List all students and classes

```
select STUDENT.*, CLASSROOMS.*
from STUDENT, CLASSROOMS
```

**Equijoin:**
List all students and classes

```
select STUDENT.*, CLASSROOMS.*
from STUDENT S, CLASSROOMS C
where S.ID = C.SID
```

**QUERY:** For each student who is taking course from instructor Fred and who has GPA > 3.0 and age > 20, list their names and addresses.
Join...

// Greater-Than Join:
Student Address > Address Student
SQL:

// Not-Equal Join:
Student Address <> Address Student
SQL:

Aggregate Functions

COUNT (attr) -- number of the values in attr
SUM(attr) -- sum of the values in attr
AVG (attr) -- average of the values in attr
MAX (attr) -- Maximum value in attr
MIN (attr) -- Minimum value in attr

List the course# and the total number of students in each course
select course#, count(SID)
from CLASSROOMS
group by course#;

List the course# and the total number of students for each course that has less than 3 students.
select course#, count(SID)
from CLASSROOMS
group by course#
Having count(SID) < 3;
**Grouping Results**

**Double Grouping**
List the average GPA of each class and group them by instructors.

```sql
select course#, Emp#, sum(GPA)/count(STUDENT.ID)
from STUDENT, CLASSROOMS
where STUDENT.ID = CLASSROOMS.SID
group by course#, Emp#;
```

- Aggregate functions are applied to the query results, not to the result construction

---

**Retrieval with Like Command**

- `'%'` any string which \( n \) characters, \( n \geq 0 \)
- `'_'` any single character.
- `x` exact sequence of string `x`.

**Result:**

List all CIS 2000 level courses.
```sql
select * from COURSE
where course# like 'CIS2%';
```

<table>
<thead>
<tr>
<th>Course#</th>
<th>Credit</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS2010</td>
<td>3.0</td>
<td>Intro to Info. Systems</td>
</tr>
</tbody>
</table>

List all introductory CIS courses.
```sql
select * from COURSE
where course# like 'CIS%' and Title like 'Intro%';
```

<table>
<thead>
<tr>
<th>Course#</th>
<th>Credit</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS2010</td>
<td>3.0</td>
<td>Intro to Info Systems</td>
</tr>
<tr>
<td>CIS3210</td>
<td>3.0</td>
<td>Intro to Program. w. VB</td>
</tr>
<tr>
<td>CIS3730</td>
<td>3.0</td>
<td>Intro to DBMS</td>
</tr>
</tbody>
</table>
Retrieval with Null Values

Queries Involve Null Values:

List all students who have either unknown address or unknown GPA.

```sql
select *
from STUDENT
where Address is null or GPA is null;
```

Nested Subqueries

```sql
SELECT attribute1 [, attrib2, attrib3, ...]
FROM relation1 [, relation2, relation3, ...]
WHERE attr [not] { in | comparison operator | exists }
( query statement(s) );
```

List names of students who are taking "CIS 2010."

```sql
select Name
from STUDENT
where ID in
( select SID
from CLASSROOMS
where course# = 'CIS2010' );
```

List names of students who are taking "CIS 2010."

```sql
select Name
from STUDENT
where ID not in
( select SID
from CLASSROOMS
where course# = 'CIS2010' );
```

List names of students who are not taking any courses.

```sql
select Name
from STUDENT
where ID not in
( select SID
from CLASSROOMS
);
**Nested Subqueries...**

**QUERY:** List all classes taken by student Jose (assuming there is only one student with name Jose).

```sql
select *
from CLASSROOMS
where SID in
  ( select ID
    from STUDENT
    where Name = 'Jose');
```

**QUERY:** For each student who is taking course from instructor Fred and who has GPA > 3.0 and age > 20, list their names and addresses.

```sql
select *
from STUDENT S
where exists
  ( select *
    from CLASSROOMS C
    where C.SID = S.ID);
```
Update Relations

UPDATE relation-name
SET attr = scalar-expression
[ , attr = scalar-expression ] ...
[ WHERE condition ] ;

Give 20% salary raise to those employees whose salary is below 20,000.
update EMPLOYEE
set salary = salary * 1.2
where salary < 20000;

// How many tuples are modified by this single statement?
// What should one do if only one tuple should be modified?

Delete Tuples

DELETE relation-name
FROM
[ WHERE condition ]

Fire all presidents.
delete from EMPLOYEE
where title like '%President%';

// How many tuples are removed by this single statement?
// What should one do if only one tuple should be removed?
**Insert New Tuples**

INSERT
INTO  \( \text{relation-name} \ [ ( \ attr \ [ , \ attr \ ] \ldots ) ] \)
VALUES \( \ ( \ value \ [ , \ value \ ] \ldots ) \);

INSERT
INTO  \( \text{relation-name} \ [ ( \ attr \ [ , \ attr \ ] \ldots ) ] \)
subquery;

```
insert into STUDENT
values ( 's6', 'Peter', 22, 'College Park', 3.8 );

insert into STUDENT ( ID, Age, Name )
values ( 's7', 36, 'Richard' );

insert into S ( s#, Name, c# )
select STUDENT.ID, Name, Course#
from STUDENT S, CLASSROOMS C
where S.ID = C.SID and GPA > 3.0;
```

---

**Query-By-Example (QBE)**

- Data displayed in tabular form
- Requests made by filling in parts of table
- Place P. (w. Microsoft, use a check) in all columns to be selected or col. 1 for all
- Place specified value in column to select matching rows
- Multiple values on same line implies "AND" condition
- Multiple values on different lines implies "OR" condition
- \( \sim \) in front of a value implies NOT...e.g. \( \sim\text{RED} = \text{NOT RED} \)
- Use same example (underlined) in same column of two (or more) tables to show a join (w. Microsoft, draw connecting relationship-line from one table to another)