Module07 – 数据库开发

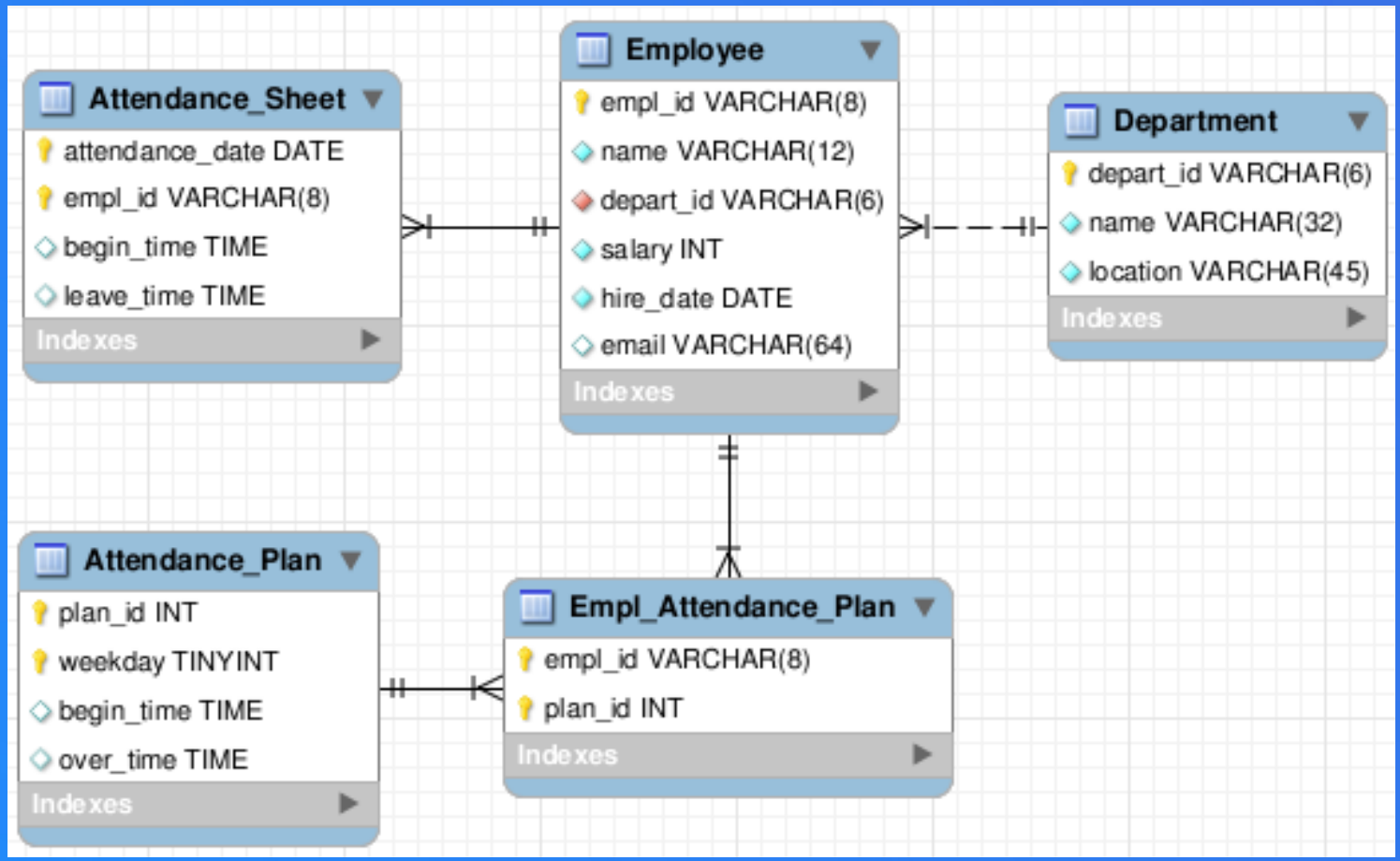
数据库开发 - 课程目标

Module06 – 数据库开发将以 Oracle10g 、 MySQL5.1 为开发环境，介绍以下几个方面的内容：

- SQL 语句：
  - DML 、 DDL 等 SQL 语句的介绍
- 存储过程、函数和触发器：
  - Oracle 和 MySQL 存储过程初步
- OTL(C++ 数据库开发接口)
  - otl v4
- 数据库建模工具简介
  - MySQL workbench 和 Sybase Power Designer

数据库开发 - 示例 ER 图

- 本次课程所用的数据库表



数据库开发 - 示例 ER 图

- 数据库表简要描述

表名	描述
Department	部门表
Employee	员工表
Attendance_Sheet	员工考勤记录表
Attendance_Plan	考勤方案表
Empl_Attendance_Plan	员工 - 考勤方案 关系表

数据库开发 - 连接到数据库

- 通过 sqlplus 连接到 Oracle 数据库实例

```
sqlplus [[user_name]/[password]]
```

- 连接到 MySQL 数据库

```
mysql -u user_name -p
password:

mysql > use database_name
```

数据库开发 - 查看数据库对象

- 查看 Oracle 数据库对象

```
--  查看当前用户拥有的表的名称
SELECT table_name FROM user_tables;

--  查看当前用户所拥有的数据库对象类型
SELECT DISTINCT object_type FROM user_objects;

--  查看当前用户所拥有的表、视图、序列等
SELECT * FROM user_catalog;
```

- 查看 MySQL 数据库对象

```
--  查看当前用户拥有的数据库名
SHOW DATABASES;

--  查看当前数据库中所有的表名
SHOW TABLES;
```

数据库开发 - 查看表结构

- describe 语句

```
describe table_name;
--  或简写
desc table_name;
```

Module07-01
数据库开发：SQL 语句

数据库开发 - SQL 语句

- SQL 语句
- Oracle PL/SQL
- MySQL Procedure
- C++ OTL
- 数据建模工具

数据库开发 - SQL 语句

- 关于 SQL 语言
  - SQL ：结构化查询语言（ Structured Query Language ），是用于数据库中的标准数据查询语言
  - 目前主流的数据库都对 SQL92 标准提供比较全面的支持

SQL 语句 - select

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

select - 选择所有行

- 选择所有列

```
--  以下语句选择表 Employee 中的所有列所有行
--  '*'  这里代表所有的列

select * from Employee;
```

- 选择指定的列

```
--  以下语句选择表 Employee 中所有行中指定的列
--  列名用 ',' 分隔

select empl_id, name, salary from Employee;
```

select - SQL 语句书写提示

- SQL 语句书写提示
  - SQL 关键字（如 select 、 from ）大小写不敏感
  - 根据创建表过程中所选择的命名方式不同，数据库对象的名称（如表、视图）和列名可以大小写敏感或不敏感
  - 在 oracle 的 sqlplus 或 mysql 的 client 交互工具中，一条 sql 语句可以分行书写，以一个结束符结束，一般是 ';'（分号）

select - 使用算术运算

- SQL 语句支持算术运算：
  - 对于数值型、日期型的列支持 + 、 - 、 * 、 / 等运算
  - 操作符的优先级： * > / > + > - （可以通过括号改变优先级）

```
-- salary * 12  （12 个月的工资）
select empl_id, name, salary * 12 from Employee;
```

```
+---------+-------------+-------------+
| empl_id | name        | salary * 12 |
+---------+-------------+-------------+
| ACC0001 | Luo jin     |       49200 |
| HRO0001 | Kong ming   |       38400 |
...
| RND0026 | Guan yu     |       70200 |
| RND0028 | Zhao yun    |       70200 |
+---------+-------------+-------------+
8 rows in set (0.00 sec)
```

select - 为列指定别名

- 为输出的列指定别名

```
-- 格式: 列名 as 别名 (as 为 SQL 关键字)
select empl_id as id, name, salary * 12 as annual_salary from
Employee;
-- 或: 列名 别名
select empl_id id, name, salary * 12 annual_salary from
Employee;
```

```
+---------+------------+---------------+
|   id    | name       | annual_salary |
+---------+------------+---------------+
| ACC0001 | Luo jin    |         49200 |
| HRO0001 | Kong ming  |         38400 |
| PUR0021 | Hui dong   |         46320 |
| RND0025 | Zhang fei  |         70200 |
| RND0026 | Guan yu    |         70200 |
| RND0028 | Zhao yun   |         70200 |
| RND0032 | Huang zhong|         72000 |
| RND0122 | Wei yan    |         62400 |
+---------+------------+---------------+
8 rows in set (0.00 sec)
```

select - 去除重复的行

- 关键字 distinct

```
select depart_id
from Employee;
```

```
+-----------+
| depart_id |
+-----------+
| ACC       |
| HRO       |
| PUR       |
| RND       |
| RND       |
| RND       |
| RND       |
| RND       |
+-----------+
8 rows in set (0.00 sec)
```

```
select distinct depart_id
from Employee;
```

```
+-----------+
| depart_id |
+-----------+
| ACC       |
| HRO       |
| PUR       |
| RND       |
+-----------+
4 rows in set (0.04 sec)
```

select - 按条件查询

- 按条件查询
  - 通过 where 子句，选择符合或不符合某些条件的行

select - where 子句

- 选择符合条件的行

```
-- 选择姓名为 Zhang fei 的员工
select * from Employee where name = 'Zhang fei';

-- 选择月薪高于 5000 的员工
select * from Employee where salary > 5000;
```

  - 关系运算操作符
    - 等于 = 、小于 < 、大于 > 、小于或等于 <= 、大于或小于 >= 、不等于 !=

select - where 子句

- 其它比较的条件
  - between ... and ... ： 界定某个区间
  - in(set) ： 判断某个值是否出现在某个 set 中
  - like ： 匹配某个字符样式
  - is null ： 判断是否为 null

```
select * from Employee where name like '%Zha%';

select * from Employee where name like '_ha%';

select * from Employee where salary between 3000 and 5000;

select * from Employee where email is null;

select * from Employee where depart_id in ('RND', 'ACC');
```

select - where 子句

- 复合条件
  - 结合 and、 or、 not 三个逻辑运算符，组成复合条件
  - 注意运算符的优先级： not > and > or
  - 可以使用括号来改变表达式的优先级

```
select * from Employee
where depart_id = 'RND' and salary > 5500;

select * from Employee
where depart_id not in ('RND', 'ACC');
```

select – order by 子句

- 使用 order by 子句来对输出的结果排序
  - 默认情况下，输出的结果按主键、升序排序
  - 可以使用 order by 子句定义排序的规则

```
select empl_id, name, salary
from Employee order by salary desc;     -- 按工资、降序排序
```

```
+---------+-------------+--------+
| empl_id | name        | salary |
+---------+-------------+--------+
| RND0032 | Huang zhong |   6000 |
| RND0025 | Zhang fei   |   5850 |
| RND0026 | Guan yu     |   5850 |
| RND0028 | Zhao yun    |   5850 |
| RND0122 | Wei yan     |   5200 |
| ACC0001 | Luo jin     |   4100 |
| PUR0021 | Hui dong    |   3860 |
| HRO0001 | Kong ming   |   3200 |
+---------+-------------+--------+
8 rows in set (0.00 sec)
```

select – order by 子句

- 使用 order by 子句来对输出的结果排序
  - 可以以多个列排序

```
--  按工资、降序和姓名、升序排序
select empl_id, name, salary
from Employee order by salary desc, name asc;
```

```
+---------+-------------+--------+
| empl_id | name        | salary |
+---------+-------------+--------+
| RND0032 | Huang zhong |   6000 |
| RND0026 | Guan yu     |   5850 |
| RND0025 | Zhang fei   |   5850 |
| RND0028 | Zhao yun    |   5850 |
| RND0122 | Wei yan     |   5200 |
| ACC0001 | Luo jin     |   4100 |
| PUR0021 | Hui dong    |   3860 |
| HRO0001 | Kong ming   |   3200 |
+---------+-------------+--------+
8 rows in set (0.00 sec)
```

select - 多表关联查询

- 常用的 join 语句
  - inner join 、 cross join
  - left join 、 right join

select – inner join 语句

- inner join 和 cross join

```
-- 也可以使用 cross join 替换
select e.empl_id, e.name, d.depart_id
from Employee e
inner join Department d
on e.depart_id = d.depart_id;
```

```
+---------+-------------+-----------+
| empl_id | name        | depart_id |
+---------+-------------+-----------+
| ACC0001 | Luo jin     | ACC       |
| HRO0001 | Kong ming   | HRO       |
| PUR0021 | Hui dong    | PUR       |
| RND0025 | Zhang fei   | RND       |
| RND0026 | Guan yu     | RND       |
| RND0028 | Zhao yun    | RND       |
| RND0032 | Huang zhong | RND       |
| RND0122 | Wei yan     | RND       |
+---------+-------------+-----------+
8 rows in set (0.00 sec)
```

select – inner join 语句

- inner join 的另外一种表达

```
select e.empl_id, e.name, d.depart_id
from Employee e, Department d
where e.depart_id = d.depart_id;
```

```
+---------+------------+-----------+
| empl_id | name       | depart_id |
+---------+------------+-----------+
| ACC0001 | Luo jin    | ACC       |
| HRO0001 | Kong ming  | HRO       |
| PUR0021 | Hui dong   | PUR       |
| RND0025 | Zhang fei  | RND       |
| RND0026 | Guan yu    | RND       |
| RND0028 | Zhao yun   | RND       |
| RND0032 | Huang zhong| RND       |
| RND0122 | Wei yan    | RND       |
+---------+------------+-----------+
8 rows in set (0.00 sec)
```

select – right/left join 语句

- right 和 left join

```
-- 也可以使用 cross join 替换
select e.empl_id, e.name, d.depart_id
from Employee e
right join Department d on e.depart_id = d.depart_id;
```

```
+---------+-------------+-----------+
| empl_id | name        | depart_id |
+---------+-------------+-----------+
| ACC0001 | Luo jin     | ACC       |
| HRO0001 | Kong ming   | HRO       |
| PUR0021 | Hui dong    | PUR       |
| RND0025 | Zhang fei   | RND       |
| RND0026 | Guan yu     | RND       |
| RND0028 | Zhao yun    | RND       |
| RND0032 | Huang zhong | RND       |
| RND0122 | Wei yan     | RND       |
| NULL    | NULL        | SAL       |
+---------+-------------+-----------+
9 rows in set (0.00 sec)
```

select – join 多表

- join 两个以上的表

```
-- 也可以使用 cross join 替换
select e.empl_id, e.name, d.depart_id
from Employee e
inner join Department d on e.depart_id = d.depart_id
inner join Attendance_Sheet a on a.empl_id = e.empl_id
where d.depart_id = 'RND'; -- 可以给定条件
```

  - left/right join 语句可以被 inner join 嵌套，反之则不行
  - 注意： join 语句可以使用 where 子句来给定条件

select - 使用组函数

- 常用的组函数
  - AVG ： 平均值
  - COUNT ： 总个数
  - MAX： 最大值
  - MIN ： 最小值
  - SUM： 和

select – 使用组函数

- 组函数的语法

```
SELECT  [column,] group_function(column), ...
FROM        table
[WHERE  condition]
[GROUP BY   column]
[ORDER BY   column];
```

```
--  如: 统计 ACC 部门的员工总数
select count(*)
from Employee
where depart_id = 'ACC';
```

select – 使用 group by 子句

- 组函数使用示例

```
select depart_id, avg(salary)
from Employee
group by depart_id;
```

```
select depart_id, count(empl_id)
from Employee
where salary > 3500
group by depart_id; -- 注意，不要颠倒 where 和 group by 子句的次序
```

```
select max(salary), depart_id
from Employee
group by depart_id;
```

select – 使用 group by 子句

- group by 多个列

```
select depart_id, count(empl_id)
from Employee
group by depart_id, salary;
```

select – 使用 group by 子句

- 错误的组函数使用语句
  - 出现在 select 列表中的列或表达式，如果不是一个组函数，则该列或表达式必须出现在 group by 子句中，如下面的例子属于不合法的 SQL 语句（注意：某些数据库系统能接受下面的语法）

```
select depart_id, count(empl_id)
from Employee
group by salary;
```

select – 使用 having 子句

- having 将影响：
  - 被归组的行
  - 使用了组函数
  - 只显示匹配 having 条件的组
- having 子句语法

```
SELECT  column, group_function
FROM        table
[WHERE  condition]
[GROUP BY   group_by_expression]
[HAVING group_condition]
[ORDER BY   column];
```

select – 使用 having 子句

- having 示例

```
select depart_id, count(empl_id)
from Employee
group by depart_id
having count(empl_id) >= 2;
```

select - 子查询

- 关于子查询
  - 所谓子查询就是将一次查询的结果用于另一次查询的条件
- 子查询语法

```
SELECT  select_list
FROM    table
WHERE   expr operator
        (SELECT select_list
         FROM   table);
```

```
-- 如: 查询工资比员工 Zhang fei 高的员工的 id 和 name
select empl_id, name
from Employee
where salary >
    (select salary
    from Employee
    where name = 'Zhang fei');
```

select - 单行子查询

- 单行子查询
  - 单行子查询只返回一行
  - 常用的单行比较操作符：
    - =、 <、 >、 <=、 >=、 !=

select - 多行子查询

- 多行子查询
  - 多行子查询只返回一行或多行
  - 常用的单行比较操作符：
    - in、 any、 all

select - 多行子查询

- 多行子查询示例

```
select empl_id, name
from Employee
where salary > any
    (select salary
    from Employee
    where depart_id = 'RND');
```

select - union 子句

- 关于 union 子句
  - UNION 用于把来自多个 SELECT 语句的结果组合到一个结果集合中。
- union 子句语法

```
SELECT ...
UNION [ALL | DISTINCT]
SELECT ...
[UNION [ALL | DISTINCT]
SELECT ...]
```

select - union 子句

- union 子句示例

```
select empl_id, salary from Employee
union
select 'total', sum(salary) from Employee;
```

```
+---------+--------+
| empl_id | salary |
+---------+--------+
| ACC0001 |   4100 |
| HRO0001 |   3840 |
| PUR0021 |   3860 |
| RND0025 |   5850 |
| RND0026 |   5850 |
| RND0028 |   5850 |
| RND0032 |   6000 |
| RND0122 |   5200 |
| total   |  40550 |
+---------+--------+
9 rows in set (0.00 sec)
```

select – case when 子句

- case when 子句语法

```
CASE
    WHEN Boolean_expression THEN result_expression
        [ ...n ]
    [
        ELSE else_result_expression
    ]
END
```

select – case when 子句

- case when 子句示例

```
-- 统计各个部门中工资高于 4000 的员工数量
select depart_id,
    sum(case when salary > 4000 then 1 else 0 end) sal4000
from Employee
group by depart_id;
```

```
+-----------+---------+
| depart_id | sal4000 |
+-----------+---------+
| ACC       |       1 |
| HRO       |       0 |
| PUR       |       0 |
| RND       |       5 |
+-----------+---------+
4 rows in set (0.00 sec)
```

select – case when 子句

- case when 子句示例 2

```
update Employee set salary =
    (case when salary < 3500 then salary + 1000
          when salary between 3500 and 3999 then salary + 800
          else salary + 600
     end); -- 按不同工资级别为员工加工资，如工资低于 3500 的加薪 1000, ...
```

```
select empl_id, salary
from Employee;
+---------+--------+
| empl_id | salary |
+---------+--------+
| ACC0001 |   4100 |
| HRO0001 |   3840 |
| PUR0021 |   3860 |
| RND0025 |   5850 |
| RND0026 |   5850 |
| RND0028 |   5850 |
| RND0032 |   6000 |
| RND0122 |   5200 |
+---------+--------+
```

更改前

```
select empl_id, salary
from Employee;
+---------+--------+
| empl_id | salary |
+---------+--------+
| ACC0001 |   4700 |
| HRO0001 |   4640 |
| PUR0021 |   4660 |
| RND0025 |   6450 |
| RND0026 |   6450 |
| RND0028 |   6450 |
| RND0032 |   6600 |
| RND0122 |   5800 |
+---------+--------+
```

更改后

SQL 语句 - insert

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

insert - 语法

- insert 语句语法

```
INSERT INTO table [(column [, column...])]
VALUES   (value [, value...]);
```

insert - 示例

- 插入的行中包括所有列

```
insert into Department
values('PRD', 'Public Relations', 'Shanghai');

-- 等同于
insert into Department
(depart_id, name, location)
values('PRD', 'Public Relations', 'Shanghai');
```

insert - 示例

- 插入的行中包括部分列

```
insert into Attendance_Plan (plan_id, weekday)
values (3, 6);
```

```
mysql> desc Attendance_Plan;
+------------+------------+------+-----+---------+-------+
| Field      | Type       | Null | Key | Default | Extra |
+------------+------------+------+-----+---------+-------+
| plan_id    | int(11)    | NO   | PRI | NULL    |       |
| weekday    | tinyint(4) | NO   | PRI | NULL    |       |
| begin_time | time       | YES  |     | NULL    |       |
| over_time  | time       | YES  |     | NULL    |       |
+------------+------------+------+-----+---------+-------+
```

SQL 语句 - delete

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

delete - 语法

- delete 语句语法

```
DELETE [FROM]   table
[WHERE    condition];
```

delete - 示例

- 删除所有行

```
delete from Empl_Attendance_Plan;
```

- 删除符合某种条件的行

```
delete from Empl_Attendance_Plan
where empl_id = 'Kong ming';
```

SQL 语句 - update

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

update - 语法

- update 语句语法

```
UPDATE  table
SET column = value [, column = value, ...]
[WHERE  condition];
```

update - 示例

- 更新一行中的特定列的值

```
update Employee set salary = salary * 1.2
where name = 'Kong ming';
```

- 更新多行中的特定列的值

```
update Employee set salary = salary + 1000
where depart_id = 'RND';
```

SQL 语句 - truncate

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

truncate - 语法

- truncate 语句语法

```
TRUNCATE [TABLE] tbl_name
```

- 与 delete 语句相比， truncate 具有以下优点：
  - 所用的事务日志空间较少。
  - delete 语句每次删除一行，并在事务日志中为所删除的每行记录一个项。 truncate table 通过释放用于存储表数据的数据页来删除数据，并且在事务日志中只记录页释放。
  - 使用的锁通常较少。
  - 当使用行锁执行 delete 语句时，将锁定表中各行以便删除。 truncate table 始终锁定表和页，而不是锁定各行。
  - 如无例外，在表中不会留有任何页。
  - 执行 delete 语句后，表仍会包含空页。

SQL 语句 - create

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

create - 数据库对象

- 数据库对象
  - 表 table
    - 基本的存储单元，由列和行组成
  - 视图 view
    - 一个逻辑上的数据库对象，由一个或多个表中的数据子集组成，可以理解为 table 的观察者
  - 索引 index
    - 为改善查询的性能而创建的数据库对象，一般将表的主键或查询性能相关的列作为索引
  - 序列 Sequence (Oracle 数据库)
    - 连续数生成器

create - 数据类型

- SQL 数据类型（部分常用类型）

类型	描述
varchar(n)	变长字符数组，最多不超过 n 个字符 （注：Oracle 中支持 varchar，但更多是以 varchar2 代替）
char(n)	n 个字符的定长字符数组
integer	整形
float	浮点型
number(p,s)	数值型，数字总长为 p，小数部分 s 位 如，number(6,2)，2103.21 符合格式，而 88821.2 不符合格式
date	日期型，如 2010-05-01 （注：Oracle 的日期包含时间部分）
time	时间型，如 10:12:45（注：Oracle10g 不支持该类型）
timestamp	时间戳，包含年、月、日、时、分、秒、以及秒的小数部分 如：2010-05-05 12:35:08.321005

create - 数据类型相关的函数

- 类型相关的函数
  - 如日期函数、字符串操作函数、数值函数、类型转换函数等，请参考以下资源：
    - Oracle SQL Functions:
      - http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/functions001.htm
    - MySQL SQL Functions
      - http://dev.mysql.com/doc/refman/5.1/en/func-op-summary-ref.html

create - 创建表

- create table 语句语法

```
CREATE TABLE [schema.]table
    (column datatype [DEFAULT expr][, ...]);
```

```
--  如创建一个新表: test_tbl
create table test_tbl (
    test_id varchar(12),
    content varchar(64)    default ' ',
    test_date date
);
```

create - 创建表

- 从子查询创建表

```
CREATE TABLE table
      [(column, column...)]
AS subquery;
```

```
-- 如创建一个新表: rnd_empls，新表有 2 个列: empl_id 和 name
create table rnd_empls
as
select empl_id, name
from Employee
where depart_id = 'RND';
```

```
-- 同上，不过这里指定新表的列名: empl_id->eid 和 name->ename
create table rnd_empls (eid varchar(8), ename varchar(16))
as
select empl_id, name
from Employee
where depart_id = 'RND';
```

create - 创建表

- 加入约束 (Constraints)
  - 可用的约束：
    - NOT NULL： 字段值不可为 NULL
    - UNIQUE ： 字段值在表范围内不可相同（重复）
    - PRIMARY KEY： 字段为表的主键
    - FOREIGN KEY： 字段为另一个表（关联表）的主键
    - CHECK： 对字段值作某些检查

create - 创建表

- 定义约束 (Constraints)

```
CREATE TABLE [schema.]table
             (column datatype [DEFAULT expr]
              [column_constraint],
              ...
              [table_constraint][,...]);
```

```
CREATE TABLE attendance_sys.Employee (
  empl_id VARCHAR(8) NOT NULL , -- NOT NULL 约束
  name VARCHAR(12) NOT NULL ,
  depart_id VARCHAR(6) NOT NULL ,
  salary INT NOT NULL ,
  hire_date DATE NOT NULL ,
  email VARCHAR(64) NULL ,
  PRIMARY KEY (empl_id) ,  -- 主键约束
  CONSTRAINT fk_Employee_Department --  外键约束
    FOREIGN KEY (depart_id)
    REFERENCES attendance_sys.Department (depart_id),
  CONSTRAINT ck_salary_min CHECK (salary > 0) -- check 约束
);
```

create - 创建表

- 使用 alter 子句追加约束 (Constraints)
  - 可以在表结构创建之后再追加某些约束

```
ALTER TABLE  table
ADD [CONSTRAINT constraint] type (column);
```

```
alter table "Employee"
   add constraint FK_EMPLOYEE_RELATIONS_DEPARTME foreign key
("Dep_depart_id")
      references "Department" ("depart_id");
```

  - 关于 alter 子句的更多介绍见后面的讲义

create - 创建表

- 查看某个表中的约束 (适用于 Oracle)

```
SELECT   constraint_name, constraint_type,
         search_condition
FROM     user_constraints
WHERE    table_name = 'Employee';
```

create - 创建视图

- 关于视图
  - 一个视图可以看作一个*虚拟表*， 也就是说，数据库里的一个 *物理上*不存在的，但是用户看起来却存在的表。 与之相比，当我们谈到一个 基本表时， 则是在物理存储中的确物理地存放着表中每一行的内容。
  - 视图没有它们自身的、物理上分开的、可区分的存储区。 实际上，系统把视图的定义（也就是说，为视图物化的应如何访问物 理上存储在基本表中内容的规则）存放在系统表里的某个地方

create - 创建视图

- create view 语法

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

```
create or replace view depart_empls
as
select d.depart_id, d.name depart_name,
    e.empl_id, e.name empl_name
from Department d, Employee e
where d.depart_id = e.depart_id and e.depart_id != 'ACC';
```

create - 创建 Sequence

- create sequence 语法 (Oracle)

```
CREATE SEQUENCE sequence
       [INCREMENT BY n]
       [START WITH n]
       [{MAXVALUE n | NOMAXVALUE}]
       [{MINVALUE n | NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE n | NOCACHE}];
```

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 1 -- 增量 1
                START WITH 1   -- 起始值 1
                MAXVALUE 9999    -- 最大值 9999
                NOCACHE
                NOCYCLE;
```

create - 创建 Sequence

- 查看 sequence 信息 (Oracle)

```
SELECT   sequence_name, min_value, max_value,
         increment_by, last_number
FROM     user_sequences;
```

- 使用 sequence
  - sequence 生成的值一般用于整型类型的主键值，可以通过：
    - seq_name.nextval 获取下一个数值
    - seq_name.currval 获取当前值

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES (dept_deptid_seq.NEXTVAL, 'Support', 2500);

SELECT dept_deptid_seq.CURRVAL FROM dual;
```

create - 创建 Sequence

- MySQL 数据库中与 Oracle Sequence 的对等机制
  - MySQL 数据库支持表的整型主键的自增来达到 Oracle Sequence 机制的效果

```
mysql> create table temp_tbl (
    id int not null auto_increment primary key,
    name varchar(32)
);

mysql> insert into temp_tbl values (null, 'Quard');

mysql> select * from temp_tbl;
+----+-------+
| id | name  |
+----+-------+
|  1 | Quard |
+----+-------+
1 row in set (0.00 sec)
```

create - 创建 Index

- create index 语法

```
CREATE INDEX index
ON table (column[, column]...);
```

```
CREATE INDEX  emp_id_idx
ON Employee(empl_id);
```

create - 创建 Index

- 什么时候需要创建 index
  - 一个列包含大量的值或一个列中含有大量的空值
  - 一个或多个列经常出现在 where 子句或 join 子句
  - 表很大，且大部分的查询动作预期返回不足 2%~4% 的行
- 什么时候不需要创建 index
  - 除不符合以上情形外，
  - 一个列的内容改变很频繁
  - 被索引的列被某个表达式引用

SQL 语句 - drop

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

drop - 删除数据库对象

- drop 语句
  - drop 语句可以删除任意数据库对象，如数据库、表、视图、索引、约束等
  - 语法

```
DROP DATABASE_OBJ OBJ_NAME;
```

```
DROP DATABASE tempdb;

DROP TABLE test_tbl;

DROP VIEW test_view;

DROP CONSTRAINTS fk_EMPLOYEE_id;

...
```

SQL 语句 - alter

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

alter - 修改数据库对象

- alter table 语句
  - alter table 语句可以修改已存在的表的定义

alter - 修改数据库对象

- alter table 语句示例

```
CREATE TABLE t1 (a INTEGER,b CHAR(10));
--  把表 t1 重新命名为 t2：
ALTER TABLE t1 RENAME t2;

--  把列 a 从 INTERGER 更改为 TINYINT NOT NULL（名称保持不变），并把列 b 从
CHAR(10) 更改为 CHAR(20)，同时把列 b 重新命名为列 c：
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);

--  添加一个新的 TIMESTAMP 列，名称为 d：
ALTER TABLE t2 ADD d TIMESTAMP;

--  在列 d 和列 a 中添加索引:
ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
ALTER TABLE t2 DROP COLUMN c; --  删除列 c

--  添加一个新的 AUTO_INCREMENT 整数列，名称为 c：
--  注意我们为 c 编制了索引（作为 PRIMARY KEY），因为 AUTO_INCREMENT 列必须编
制索引。同时我们定义 c 为 NOT NULL，因为主键列不能为 NULL。
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
    ADD PRIMARY KEY (c);
```

SQL 语句 - alter

- SQL 语句：
  - DML(数据操作语句)
    - select
    - insert
    - delete
    - update
    - truncate
  - DDL(数据定义语句)
    - create
    - drop
    - alter
  - 事务

事务 - 概要

- 关于事务 (Transaction)
  - 事务（ transaction ）是由一系列操作序列构成的程序执行单元，这些操作要么都做，要么都不做，是一个不可分割的工作单位。

事务 - 事务的基本性质

- 事务的 4 个基本性质（ ACID ）
  - 原子性（ Atomicity ）
    - 事务的原子性是指事务中包含的所有操作要么全做，要么全不做（ all or none ）。
  - 一致性（ Consistency ）
    - 在事务开始以前，数据库处于一致性的状态，事务结束后，数据库也必须处于一致性状态。
      拿银行转账来说，一致性要求事务的执行不应改变 A 、 B 两个账户的金额总和。如果没有这种一致性要求，转账过程中就会发生钱无中生有，或者不翼而飞的现象。事务应该把数据库从一个一致性状态转换到另外一个一致性状态。

事务 - 事务的基本性质

- 事务的 4 个基本性质（ ACID ）（续）
  - 隔离性（ Isolation ）
    - 事务隔离性要求系统必须保证事务不受其他并发执行的事务的影响，也即要达到这样一种效果：对于任何一对事务 T1 和 T2，在事务 T1 看来，T2 要么在 T1 开始之前已经结束，要么在 T1 完成之后才开始执行。这样，每个事务都感觉不到系统中有其他事务在并发地执行。
  - 持久性（ Durability ）
    - 一个事务一旦成功完成，它对数据库的改变必须是永久的，即使是在系统遇到故障的情况下也不会丢失。数据的重要性决定了事务持久性的重要性。

事务 - 事务的隔离级别

- 关于事务的隔离级别 (transaction isolation levels)
  - SQL 标准用三个必须在并行的事务之间避免的现象定义了四个级别的事务隔离。 这些不希望发生的现象是：
    - 脏读（ Dirty Read ）
      一个事务读取了另一个未提交的并行事务写的数据。
    - 不可重复读（ NonRepeatable Read ）
      一个事务重新读取前面读取过的数据， 发现该数据已经被另一个已提交的事务修改过。
    - 幻读（ Phantom Read ）
      一个事务重新执行一个查询，返回符合查询条件的行， 发现这些行因为其它最近提交的事务而发生了改变。

事务 - 事务的隔离级别

- 4 个隔离级别

  以下是 4 个事务隔离级别的描述，级别越高、数据越安全可靠，但数据库操作的开销也越大。

  - 读未提交（ Read uncommitted ）
    - 这是最低的隔离等级，允许其他事务看到没有提交的数据。这种等级会导致脏读（ Dirty Read ）。
  - 读已提交（ Read committed ）
    - 被读取的数据可以被其他事务修改。这样就可能导致不可重复读。也就是说，事务的读取数据的时候获取读锁，但是读完之后立即释放（不需要等到事务结束），而写锁则是事务提交之后才释放。释放读锁之后，就可能被其他事务修改数据。该等级也是多数数据库管理系统的默认隔离等级。

事务 - 事务的隔离级别

- 4 个隔离级别（续）
  - 可重复读（ Repeatable read ）
    - 所有被 Select 获取的数据都不能被修改，这样就可以避免一个事务前后读取数据不一致的情况。但是却没有办法控制幻读，因为这个时候其他事务不能更改所选的数据，但是可以增加数据，因为前一个事务没有范围锁。
  - 可串行化（ Serializable ）
    - 所有事务都一个接一个地串行执行，这样可以避免幻读（ phantom reads ）。对于基于锁来实现并发控制的数据库来说，串行化要求在执行范围查询（如选取工资在 4500 到 6000 之间的用户）的时候，需要获取范围锁（ range lock ）。如果不是基于锁实现并发控制的数据库，则检查到有违反串行操作的事务时，需要回滚该事务。

事务 - 事务的隔离级别

- 隔离级别汇总

事务隔离级别（从低到高）	脏读 (dirty reads)	不可重复读 (non-repeatable reads)	幻读 (phantom reads)
读未提交 (Read uncommitted)	可能	可能	可能
读已提交 (Read committed)	不可能	可能	可能
可重复读 (Repeatable read)	不可能	不可能	可能
可串行化 (Serializable)	不可能	不可能	不可能

事务 - 事务的产生和结束

- 数据库事务有以下情形之一构成：
  - 组成对数据一系列更改的 DML 语句
  - DDL 语句（如 CREATE TABLE 语句等）
  - DCL 语句（如 GRANT 语句等）
- 数据库事务的开始和结束
  - 第一个 DML 语句的执行
  - 遇到以下事务之一，结束事务
    - 执行 COMMIT 或 ROLLBACK 语句
    - 执行一条 DCL 或 DDL 语句（自动 COMMIT ）
    - 正常退出 sqlplus 或 mysql client
    - 数据库系统关闭

事务 - 回滚事务

- 回滚事务： ROLLBACK

```
set autocommit=false;

update Employee set salary = salary + 1005
where salary < 5000; --#1

delete from Employee where salary = 5705; --#2

rollback; --#3
```

```
+---------+--------+
| empl_id | salary |
+---------+--------+
| ACC0001 |   5705 |
| HRO0001 |   5645 |
| PUR0021 |   5665 |
+---------+--------+
```

#1

```
+---------+--------+
| empl_id | salary |
+---------+--------+
| HRO0001 |   5645 |
| PUR0021 |   5665 |
+---------+--------+
```

#2

```
+---------+--------+
| empl_id | salary |
+---------+--------+
| ACC0001 |   4700 |
| HRO0001 |   4640 |
| PUR0021 |   4660 |
+---------+--------+
```

#3

事务 - 提交事务

- 提交事务： COMMIT

```
set autocommit=false;

update Employee set salary = salary + 1005
where salary < 5000; --#1

delete from Employee where salary = 5705; --#2

commit; --#3
```

SQL 语句 - 小结

- 本小结简要介绍了主要的 DML 语句和 DDL 语句
- 由于 Oracle 和 MySQL（和其它）数据库系统对 SQL 语言的标准支持的程度不尽相同，本小结中尽量取 Oracle 和 MySQL 均支持的语法
- 如果针对特定的数据库开发，则有必要了解该数据库厂商的 SQL 扩展
- Oracle 和 MySQL 各自的字符串、日期、数据转换的函数使用十分频繁，有必要熟练掌握常用的内置函数
- 在数据库开发过程中， select 语句的性能往往直接制约应用程序的性能，所以对 select 语句的优化的重要性远比其它语句来得重要