

1. 数据库的三个范式

- a) **1NF**: 强调的是列的原子性, 即列不能够再分成其他几列。
- b) **2NF**: 首先是 **1NF**, 另外包含两部分内容, 一是表必须有一个主键; 二是没有包含在主键中的列必须完全依赖于主键, 而不能只依赖于主键的一部分。(单一主键或者所有列都是主键的满足 **1NF** 的数据库一定满足 **2NF**)
- c) **3NF**: 首先是 **2NF**, 另外非主键列必须直接依赖于主键, 不能存在传递依赖。即不能存在: 非主键列 **A** 依赖于非主键列 **B**, 非主键列 **B** 依赖于主键的情况。

2. 创建外键

```
CREATE TABLE interests (
    int _id INT NOT NULL AUTO _ INCREMENT PRI
    interest VARCHAR(50) NOT NULL,
    contact _id INT NOT NULL,
    CONSTRAINT my _ contacts _ contact _id
    FOREIGN KEY (contact _id)
    REFERENCES my _ contacts (contact _id)
```

3. 进阶 SELECT

1. CASE 语句

```
UPDATE my_table
SET new_column =
CASE
    → WHEN column1 = somevalue1
    → THEN newvalue1
    → WHEN column2 = somevalue2
    → THEN newvalue2
    ELSE newvalue3
END;
```

任何不符合上述条件的记录

2. ORDER BY 语句（可以在后面加 DESC）

多列排序: ORDER BY 后面依次加入要排序的行

3. 统计函数: SUM, MAX, MIN, AVG 等

4. 分组关键词：GROUP BY。

5. 统计行数 COUNT()

6. 只选择不重复的: DISTINCT 数据

7. 限制结果的数量: LIMIT (编号从 1 开始) 或者 TOP (可以选择前百分之几)

一个参数：结果集数量

- 两个参数：从第几个开始，结果集数量
8. 模糊查询：LIKE
%：匹配多个字符
?：匹配一个字符
 9. 字符串限制的查询：SUB_STRING()....
 10. 数据结构转换：SELECT CASE([column_name] AS TYPE)
4. 想新建表，然后把查询到的东西插入新表，应该怎么写

1. CREATE TABLE, 然后利用 SELECT 进行 INSERT

我们已经知道这种方式了！首先创建（CREATE）profession表，然后填入第345页上的SELECT的查询结果。

```
CREATE TABLE profession
(
  id INT(11) NOT NULL AUTO _ INCREMENT PRIMARY KEY,
  profession varchar(20)
);

INSERT INTO profession (profession)
SELECT profession FROM my_contacts
GROUP BY profession
ORDER BY profession;
```

创建带有主键列的profession表，用VARCHAR类型的列存储职业。

现在以SELECT的查询结果填满profession表的profession列。

1.

2. 利用 SELECT 进行 CREATE TABLE, 然后 ALTER 以添加主键

第二种方式：利用SELECT从my_contacts表的职业列抓出来的数据创建新的profession表，再用 ALTER 修改新表并添加（ADD）主键字段。

```
CREATE TABLE profession AS
SELECT profession FROM my_contacts
GROUP BY profession
ORDER BY profession;

ALTER TABLE profession
ADD COLUMN id INT NOT NULL AUTO _ INCREMENT FIRST,
ADD PRIMARY KEY (id);
```

创建只有表，并保存结果.....

.....然后用 ALTER 改表以添加主键！

2.

```
CREATE TABLE profession
(
  id INT(11) NOT NULL AUTO _ INCREMENT PRIMARY KEY,
  profession varchar(20)
) AS
SELECT profession FROM my_contacts
GROUP BY profession
ORDER BY profession;
```

满 profes

3.

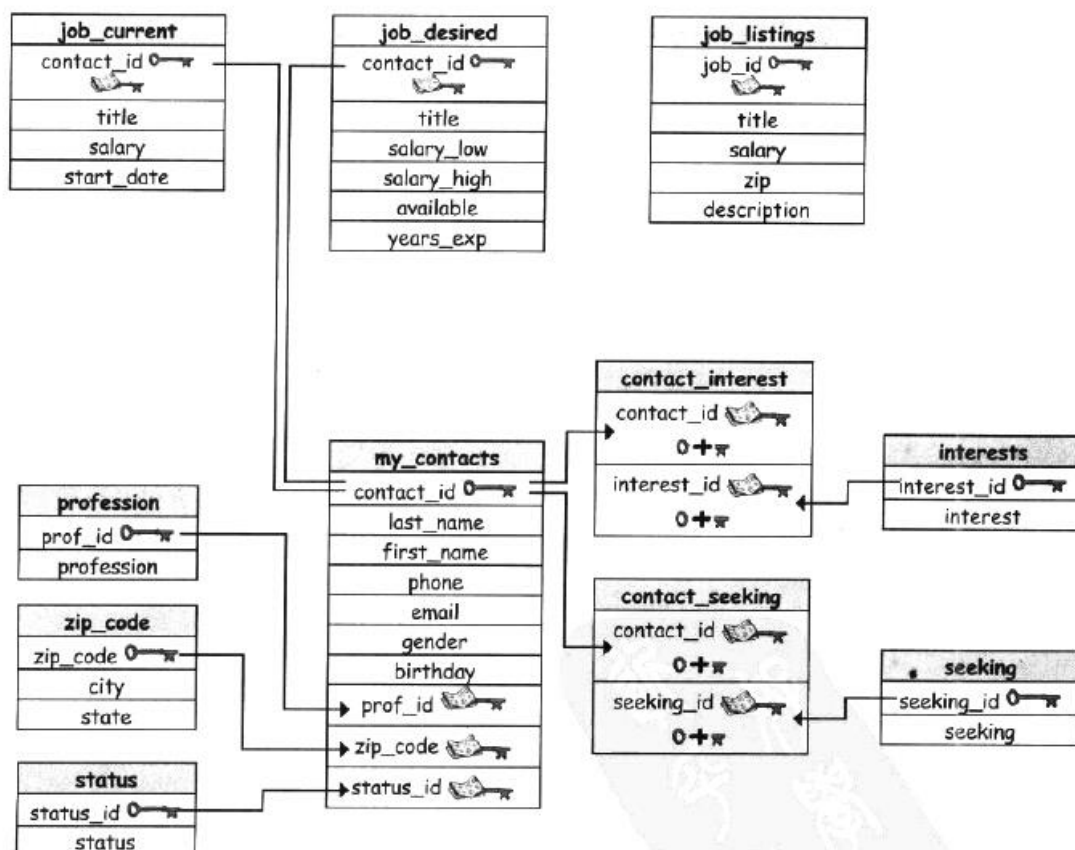
3. INNER JOIN : 连接两张表

```
SELECT mc.email, p.profession
FROM my_contacts AS mc
INNER JOIN
profession AS p
WHERE mc.contact_id=p.prof_id
```

```
SELECT mc.last_name, mc.first_name, s.status
FROM my_contacts AS mc
INNER JOIN
status AS s
WHERE mc.contact_id=s.status_id
```

```
SELECT cs.cid, s.seeking
FROM cs
NATURAL JOIN
S
```

4. 自然联接: NATURAL JOIN (用于联接的两张表里有相同名称的行)
5. 子查询: 在查询中嵌套查询



```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE jc.title IN (SELECT title FROM job_listings);
```

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
INNER JOIN job_listings jl
ON jc.title = jl.title;
```

可使用 INNER JOIN
替代包含子查询的
WHERE 子句。

1. 返回多行数据: IN 或者 NOT IN
2. 返回一行数据: =
3. 判断是否大于/小于所有的数据: ALL

```
SELECT name, rating FROM restaurant_ratings
WHERE rating > ALL
(SELECT rating FROM restaurant_ratings
WHERE rating > 3 AND rating < 9);
```

4. 判断大于返回值中最小值或者小于返回值中最大值: ANY/SOME
5. 经典例子: 在我的通讯录中, 谁赚得钱最多?

```
SELECT mc.first_name, mc.last_name, jc.salary
FROM my_contacts AS mc NATURAL JOIN job_current AS jc
WHERE jc.salary =
(SELECT MAX(jc.salary) FROM job_current jc);
```

```
SELECT mc.first_name, mc.last_name,
(SELECT state
FROM zip_code
WHERE mc.zip_code = zip_code) AS state
FROM my_contacts mc;
```

6. 非关联子查询: 内层查询不需要来自外层就可以独自运行
7. 例子: 列出每个邮编覆盖的地区收入最高的人

```
SELECT last_name, first_name FROM my_contacts
WHERE zip_code IN (SELECT mc.zip_code FROM my_contacts mc
NATURAL JOIN job_current jc
WHERE jc.salary = (SELECT MAX(salary) FROM job_current));
```

8. 关联子查询

```
SELECT mc.first_name, mc.last_name
FROM my_contacts AS mc
WHERE
3 = (
SELECT COUNT(*) FROM contact_interest
WHERE contact_id = mc.contact_id
);
```

my_contacts 的别名在
层查询中。

外
层

1. 常用法：找出外层查找中，不存在在关联表里的记录

例子：找出所有在 my_contacts 里但是不在 job_current 里的人

```
SELECT mc.first _ name firstname, mc.last _ name
FROM my _ contacts mc
WHERE NOT EXISTS
  (SELECT * FROM job _ current jc
   WHERE mc.contact _ id = jc.contact _ id );
```

NOT EXISTS 负责从 my_contacts 中找出，他们并未列在 job_current 中

7. 外联接

1. 更注重两张表之间的关系。

1. LEFT OUTER JOIN【左外联接】接受左边表的所有行，并用这些行和右表的行进行匹配，特别适合一对多关系

2. 例一

```
SELECT g.girl, t.toy
FROM girls g
LEFT OUTER JOIN toys t
ON g.toy _ id = t.toy _ id;
```

位于LEFT OUTER JOIN 以girls是左表

与内连接不同之处：外连接一定会提供数据行，无论该行是否能在另一个表中找到匹配行（没有则返回 NULL，有多行则返回多行，有 NULL 的列对应的一定是右表）

3. 右外连接=左外连接+交换表的位置
4. 自引用外键：主键在同一张表里做外键
5. 自联接：与自身进行外联接

```
SELECT c1.name, c2.name AS boss
FROM clown _ info c1
INNER JOIN clown _ info c2
ON c1.boss _ id = c2.id;
```

```
SELECT c1.name,
  (SELECT name FROM clown _ info
   WHERE c1.boss _ id = id) AS boss
FROM clown _ info c1;
```

（子查询版）

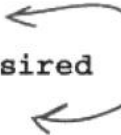
8. 联合查询

1. 例一：查询三张表

```

SELECT title FROM job _ current
UNION
SELECT title FROM job _ desired
UNION
SELECT title FROM job _ listings;

```



2. SQL 联合规则

1. 每个 SELECT 语句中列的数量必须一致
2. 每个 SELECT 语句包含的表达式与统计函数必须相同
3. SELECT 语句的顺序不重要
4. SQL 默认清除重复值（使用 UNION ALL 查看重复值）
5. 列的数据类型必须相同或者可以互相转换
3. INTERCEPT: 只返回两个查询共有的记录（不在 MYSQL 里）
4. EXCEPT: 只返回在第一个查询中有的记录（不在 MYSQL 里）

9. 联接 VS 子查询


10. 检查数据 CHECK

1. 之前学过的: NOT NULL, UNIQUE, PRIMARY KEY, ...
2. CHECK: 只允许输入某些特定的值, 可以使用所有 WHERE 语句能使用的词（不在 MYSQL 里, 也是惨）

```

CREATE TABLE piggy _ bank
(
    id INT AUTO _ INCREMENT NOT NULL PRIMARY KEY,
    coin CHAR(1) CHECK (coin IN ('P','N','D','Q'))
)
ALTER TABLE my _ contacts
ADD CONSTRAINT CHECK gender IN ('M','F');

```



11. 视图 VIEW

1. 格式:

CREATE VIEW [name] AS

[一大堆查询]

DROP VIEW [name]

2. 查看 VIEW: SELECT * FROM [name]

3. 本质: 虚拟表, 但不会保存在数据库

4. 使用视图的原因:

- a) 视图把复杂的查询简化成了虚拟表
- b) 即使数据库变化, 也不会破坏依赖数据库的程序
- c) 视图可以向使用者隐藏数据

5. 可以使用视图做 UPDATE, DELETE, DROP 等, 但是一般传统方法更好

6. WITH CHECK OPTION: INSERT 和 DELETE 会被检查是否符合 WITH CHECK OPTION 之前的 WHERE 子句, 因此也可以模拟 CHECK

12. TRANCATION

1. 判断一组操作是否是 TRANCATION 的条件 (ACID)

A: atomic 原子性, 要么都完成, 要么一个也不做

- C: consistency 一致性, 必须保证数据一致
- I: isolation 隔离性: 每次操作都应该看到一致的数据库
- D: durability 持久性: 数据库应正确储存数据
- 2. 存储事务必须是 BDB 或者 InnoDB 两种引擎。
改变引擎: ALTER TABLE [name] TYPE=InnoDB
- 4. 使用事务
 - a) START TRANSACTION (开始事务, 所有操作记录进日志)
 - b)
 - c) COMMIT (提交修改) / ROLLBACK (回滚)
- 13. 账号管理
 - 1. root 账号: 一定要设置密码

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('b4dc10wnZ');
```

- 2. 创建用户 (新创建的用户没有任何权限)

```
CREATE USER elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';

GRANT SELECT ON
clown_info
TO elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';
```

- 2.5. 确认当前账号

```
SELECT CURRENT_USER;
```

- 3. GRANT 语句

- 1. 简单的授予权限

```
GRANT [SELECT/DROP/ALTER/DELETE/UPDATE/ALL]
ON [(database_name.)table_name/database_name.*(DATABASE 内所有表)/*.*(所有表)]
TO [user_name] (不能使用 ALL, 必须指定用户名)
```

- 2. 只授予某一列的权限

例子: GRANT SELECT(email, tel) ON my_contacts TO Jack;
除 SELECT 以外, 几乎一切只针对列的权限都不太有用

- 3. 允许 user 把自己所有的权限 GRANT 给别人 (只能给操作权限, 不能把 WITH GRANT OPTION 的权限再给出去)
(GRANT STATEMENT) WITH GRANT OPTION

- 4. 撤销权限

```
REVOKE [SELECT/DROP/ALTER/DELETE/UPDATE/ALL]
ON [(database_name.)table_name/database_name.*(DATABASE 内所有表)]
FROM [user_name] (不能使用 ALL, 必须指定用户名)
```

- 5. 不允许 user 把自己的权限再给出去

```
REVOKE GRANT OPTION ON [SELECT/DROP/ALTER/DELETE/UPDATE/ALL]
```


ON [(database_name.)table_name/database.*(DATABASE 内所有表)]

FROM [user_name] (不能使用 ALL, 必须指定用户名)

- GRANT OPTION 被撤销后, 用被撤销用户那里得到的权限也会一并被撤销 (默认)

- 可以精确指定撤销范围

- REVOKE GRANT OPTION ... FROM [] CASCADE
(一起撤销)

- REVOKE GRANT OPTION ... FROM [] RESTRICT
(如果被撤销用户已经把权限授予过别人, 则会返回错误, 权限保留)

4. 不要使用共享账号, 涉及到多人多层次权限时, 应使用角色 (ROLE) 实现

1. 创建角色

CREATE ROLE [role_name] (不在 MySQL 中)

DROP ROLE [role_name]

2. 给予权限: 完全和单人用户一样

3. 把角色赋予用户

GRANT [role_name] TO [user_name]

REVOKE [role_name] FROM [user_name]

4. 把授予角色的权力赋予用户

GRANT [role_name] TO [user_name] WITH ADMIN OPTION

REVOKE ADMIN OPTION ON [role_name] FROM [user_name]

[RESTRICT/CASCADE]

5. 使用中的角色也可以被卸除, 同时被赋予角色的用户丧失角色权限

6. 用户可身兼多个角色, 但是否定性的授予优先

14. 其他类型

	有符号整数	无符号整数
SMALLINT	-32768 ~ 32767	0 ~ 65535
BIGINT	-9223372036854775808 ~ 9223372036854775807	0 ~ 18446744073709551615

MySQL 还进一步细分出下列类型:

	有符号整数	无符号整数
TINYINT	-128 ~ 127	0 ~ 255
MEDIUMINT	-8388608 ~ 8388607	0 ~ 16777215

15. 临时表: CREATE TEMPORARY TABLE

使用临时表的原因:

1. 保存中间结果
2. 捕捉某个时刻表的状态
3. 帮助整理数据结构

16. 其他数字函数

数字函数		功能说明
ABS(x)	返回 x 的绝对值	
	查询	结果
	SELECT ABS(-23);	23
ACOS(x)	返回 x 的反余弦值	
	SELECT ACOS(0);	1.5707963267949
ASIN()	返回 x 与 y 的反正弦值	
	SELECT ASIN(0.1);	0.10016742116156
ATAN(x,y)	返回 x 与 y 的反正切值	
	SELECT ATAN(-2,2);	-0.78539816339745
CEIL(x)	返回大于等于 x 的最小整数。返回值为 BIGINT	
	SELECT CEIL(1.32);	2
COS(x)	返回 x 的余弦值。以弧度计算	
	SELECT COS(1);	0.54030230586814
COT(x)	返回 x 的余切值	
	SELECT COT(12);	-1.5726734063977
EXP(x)	返回 e 的 x 次方	
	SELECT EXP(-2);	0.13533528323661
FLOOR(x)	返回小于等于 x 的最大整数	
	SELECT FLOOR(1.32);	1
FORMAT(x,y)	转换 x 为文本字符串并四舍五入至 y 指定的位数。	
	SELECT FORMAT(3452100.50,2);	3,452,100.50
LN(x)	返回 x 的自然对数	
	SELECT LN(2);	0.69314718055995
LOG(x) 与 LOG(x,y)	返回 x 的自然对数，若有两个参数，则以 x 为基数，返回 y 的对数	
	SELECT LOG(2);	0.69314718055995
	SELECT LOG(2,65536);	16

数字函数	功能说明	
MOD(x,y)	返回 x 除以 y 的余数	
	查询	结果
	SELECT MOD(249,10);	9
PI()	返回 pi	
	SELECT PI();	3.141593
POWER(x,y)	返回 x 的 y 次方值	
	SELECT POW(3,2);	9
RADIANS(x)	返回 x 从角度转换成弧度的值	
	SELECT RADIANS(45);	0.78539816339745
RAND()	返回随机浮点数	
	SELECT RAND();	0.84655920681223
ROUND(x)	返回 x 四舍五入后最接近的整数	
	SELECT ROUND(1.34);	1
	SELECT ROUND(-1.34);	-1
ROUND(x,y)	以 y 指定的小数位数对 x 四舍五入	
	SELECT ROUND(1.465, 1);	1.5
	SELECT ROUND(1.465, 0);	1
	SELECT ROUND(28.367, -1);	30
SIGN(x)	当 x 是正数时, 返回 1; x 是 0 时, 返回 0; x 是负数时, 返回 -1	
	SELECT SIGN(-23);	-1
SIN(x)	返回 x 的正弦值	
	SELECT SIN(PI());	1.2246063538224e-16
SQRT(x)	返回 x 的平方根	
	SELECT SQRT(100);	10
TAN(x)	返回 x 的正切值	
	SELECT TAN(PI());	-1.2246063538224e-16
TRUNCATE(x,y)	返回 x 截断至 y 指定的小数位数后的值	
	SELECT TRUNCATE(8.923,1);	8.9

17. 索引: 索引占用额外空间, 但是加在常用的列上会加快搜索速度

```
ALTER TABLE my _ contacts
ADD INDEX (last _ name);
```