Week5 - sql

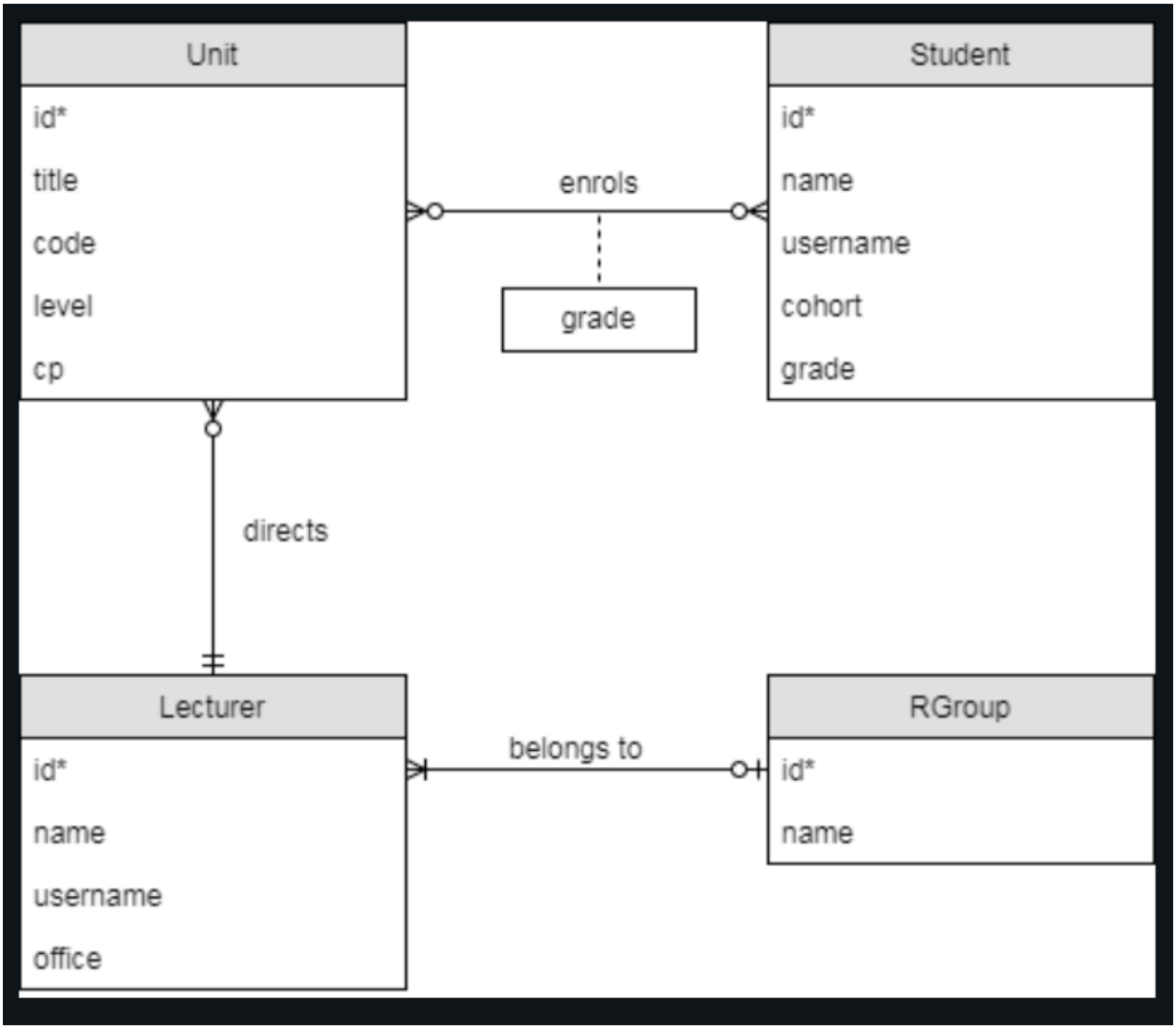
SLIDES

ER_DIAGRAM

- when exam using key.

student have name and number. student → entity , name & number → attributes.

Different key in ER diagram



o 代表可以为空; I 代表不能为空 ；I 代表 对一，三个角的那个代表对多

图示上的图只确定了大致关系，还没有确定是具体哪个键对哪个键。

SQL_BASICS

```
# create table
CREATE TABLE IF NOT EXISTS tb_name(
 `name` 类型(长度) NOT NULL,
 `number` 类型(长度) NOT NULL
);
```

```
# drop table
DROP TABLE IF EXISTS tb_name;

# add data in table
INSERT INTO tb_name(col_id1, col_id2)
 -> VALUES("data1","data2"); # 行和数据必须一一对应，要是添加所有数据，

# select rows from table
SELECT * FROM tb_name
 -> LIMIT number; # 限制显示行数

# two tables combine into one
# 并不生成新的表，只是根据条件生成新的结果集，方便在特定条件下的查询
SELECT * FROM tb_name1 JOIN tb_name2
 -> on tb_name1.col1 = tb_name2.col1  # join的条件
 -> LIMIT number;

# reducing columns
SELECT tb_name1.needed_col, tb_name2.needed_col
-> FROM tb_name1 JOIN tb_name2
-> ON tb_name1.col = tb_name2.col # 一般这里的col相同？
-> LIMIT number;
# 这样就只会剩下 tb_name1.needed_col, tb_name2.needed_col 这两行

# rename columns
SELECT tb_name1.col AS new_col_name,
    -> tb_name2.col AS new_col_name
    -> FROM tb_name1 JOIN tb_name2
  -> ON tb_name1.col = tb_name2.col
  -> LIMIT number;

# where加载特定条件
SELECT album.title AS album,
artist.name AS artist
FROM album JOIN artist
ON album.artistid = artist.artistid
```

```
WHERE album LIKE '%Rock%' # 只对包含'Rock'字符串的专辑筛选后,返回满足
LIMIT 5;
# 在SQL语言中，%是通配符，表示任意长度的任意字符。
# 在这个查询中，'Rock'被包裹在%符号中，意味着我们正在寻找包含
# 'Rock'字符串的任何album.title，而不仅仅是精确匹配'Rock'的album.titl

# DISTINCT 去重 - 只显示独一无二的
SELECT DISTINCT artist.name AS artist
FROM album JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
LIMIT 5;

# count - how many - 计数 -
# group by col_name - 通过特定条件分组
SELECT artist.name AS artist,
COUNT(album.title) as albums # count() 统计指定的非空列数值并返回- 忽
FROM album JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
GROUP BY artist # 通过artist分组(相同artist在同一个分组中)
LIMIT 5;

# ordered - 排序 默认升序 DESC 降序
SELECT artist.name AS artist,
COUNT(album.title) as albums
FROM album JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
GROUP BY artist
ORDER BY albums DESC
LIMIT 5;
```

Database Normal Forms（数据库范式）

数据库设计中有五个主要的范式，通常简称为1NF、2NF、3NF、BCNF和4NF。每个范式都提供了一组规则，用于规范化关系数据库模式，以确保数据的一致性、完整性和最小化冗余。

1. 第一范式（1NF）：确保每个属性都是原子的，即每个属性都不可再分。Each column shall contain one(and only one) value.
2. 第二范式（2NF）：确保每个非主属性完全依赖于候选键，消除部分依赖。Every non-key attribute is fully dependent on the key.
3. 第三范式（3NF）：确保每个非主属性不传递依赖于候选键，消除传递依赖。Every non-key attribute must provide a fact about the key, the whole key and nothing but the key.
4. 巴斯-科德范式（BCNF）：确保每个非平凡函数依赖都是由候选键决定的，消除主属性决定非候选键属性的函数依赖。
5. 第四范式（4NF）：确保一个关系模式中不存在多值依赖，即消除多值依赖。

这些范式按照顺序逐步提高数据库设计的规范化程度，同时减少了数据冗余和插入、更新和删除异常的可能性。

- Each column shall contain one (and only one) value
- Every non-key arttribute is fully dependent on the key

In conclusion

换句话说，2NF要求一个关系模式中的每个非主属性都必须完全依赖于关系模式的候选键，而不是部分依赖。这意味着每个非主属性的值必须取决于关系模式中的整个候选键，而不是其中的一部分。这样可以确保数据的一致性和完整性，避免了数据冗余和异常。

完全依赖 vs 部分依赖

在关系数据库设计中，"完全依赖"和"部分依赖"是描述属性对候选键的依赖程度的概念。

1. 完全依赖（Full Dependency）：一个属性完全依赖于关系模式的候选键，当且仅当它不依赖于候选键的任何真子集时。换句话说，如果一个属性的值由关系模式的整个候选键唯一确定，则该属性完全依赖于候选键。
2. 部分依赖（Partial Dependency）：一个属性部分依赖于关系模式的候选键，当且仅当它依赖于候选键的某个真子集时。换句话说，如果一个属性的值由关系模式的部分候选键（而不是整个候选键）确定，则该属性部分依赖于候选键。

举个例子来说明：

假设我们有一个关系模式R(A, B, C, D)，其中{A, B}是候选键。如果属性C的值**只取决于属性A**，而不取决于属性B，那么C**部分依赖**于候选键{A, B}。但是如果**属性C的值同时取决于属性A和属性B的组合，而不是单独的A或B，那么C完全依赖于候选键{A, B}**。

因此，部分依赖意味着属性的值仅依赖于候选键的部分，而完全依赖意味着属性的值完全依赖于整个候选键。在关系数据库设计中，我们通常希望消除部分依赖，以确保关系模式满足第二范式（2NF）的要求。

SQL_ADVANCED

NULL - means attribute missing - terrible!

解决方法： declare everything as NOT NULL

1. INNER JOIN：INNER JOIN返回两个表中符合连接条件的行，即返回两个表中共同满足连接条件的行。如果一个表中的行没有在另一个表中找到匹配的行，则这些行将被排除在结果集之外。
2. LEFT JOIN：LEFT JOIN返回左边表中的所有行，以及右边表中符合连接条件的行。如果右边表中没有与左边表中的行匹配的行，则将返回NULL值。左连接保留了左边表中的所有行，即使在右边表中没有匹配的行也是如此。
3. RIGHT JOIN：RIGHT JOIN返回右边表中的所有行，以及左边表中符合连接条件的行。如果左边表中没有与右边表中的行匹配的行，则将返回NULL值。右连接保留了右边表中的所有行，即使在左边表中没有匹配的行也是如此。

```
SELECT * FROM fruit WHERE fruit is NOT NULL;

# 默认的join - inner join - 不会返回NULL值
# 高级的join - left join & right join
# left join & right join - sometimes we're okry with database st
# left join - 左表所有行 + 右表匹配行 + 空行NULL
SELECT person, fruit.fruit, dish
FROM fruit
LEFT JOIN recipes
ON fruit.fruit = recipes.fruit;
```

```
# right join - 右表所有行 + 左表匹配行 + 空行NULL
SELECT recipes.fruit, dish, person
FROM fruit
RIGHT JOIN recipes
ON fruit.fruit = recipes.fruit;
# full join - 连接所有
SELECT * FROM fruit
FULL OUTER NATURAL JOIN recipes;

# count() - 非NULL计数 | avg() - 平均 | sum() - 总和
# 除法不准确的时候，可以给count* 1.0 -> 让结果转化为浮点数

# 计算标准差 standard deviation
SELECT SQRT(AVG(Deviation)) AS STDDEV
FROM (
SELECT Fruit, Stars, Mean,
(Stars-Mean)*(Stars-Mean) AS Deviation
FROM ranking JOIN (
SELECT AVG(stars) AS Mean
FROM ranking
)
WHERE stars IS NOT NULL
);
# 用到了：
# 1. 子查询（Subquery）：查询中嵌套了一个子查询，
# 用于计算星级（Stars）与平均值（Mean）之间的偏差（Deviation）。
# 2. JOIN：使用了JOIN操作，将ranking表与子查询的结果进行连接。
# 3. WHERE子句：在子查询中使用了WHERE子句，以过滤掉stars列中的NULL值。
# 4. AVG()函数：用于计算平均值。
#5. SQRT()函数：用于计算平方根，以得到标准差的值。
```

JDBC - JAVA DATABASE CONNECTIVITY

jdbc-access SQL from Java ***library:*** java.sql & javax.sql packages

Always using prepared statements: In order to prevent a **horrible** vulnerability called an **injection attack (**Shellshock vulnerability)
What a prepared statement does is
ensure that the things you add are what you say they are ; Suppose you do the something **similar for the login code**:

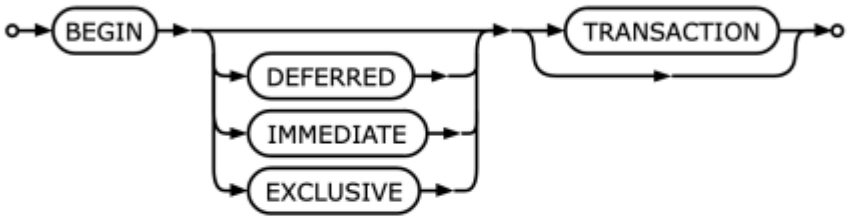
- Make sure you load the **right drive**r 驱动
- Catch SQLExceptions 解决错误
- ***Use prepared statements and transactions to prevent errors***
- And an ORM like Hibernate if you like. | objection映射工具

Transcation(事物) in JDBC：

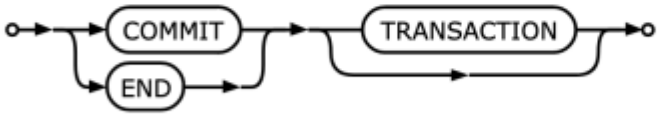
1. Start a new transaction
2. Do your work
3. Commit to it when done
4. Rollback if an error occurs

(Oh, and BTW SQLite also can do transactions in SQL)

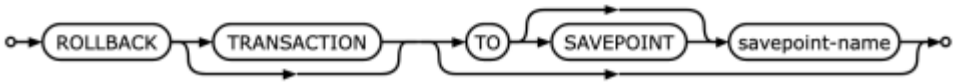
begin-stmt: hide



commit-stmt: hide



rollback-stmt: hide



Q11. What normal form is Bob's soup ranking table in?

Flavour	Ranking	Notes
Tomato	8	
Orange	3	Surprising and fruity
Chocolate Orange	0	Disgusting!

A. 3NF

B. No normal form

C. 1NF

D. 2NF

用这张图举例范式，1NF满足，因为每列不可再分（不可能有新的竖列）

2NF满足，因为在这里Flavour是主键，完全被依赖

只满足2NF → 除了主键，其他键之间还能有联系。

3NF满足，因为只能从Ranking本身，或者Notes本身推出Flavour，Ranking和Notes本身没互相推导的关系（这NF真的难懂

Q12. Bob wants to find what their mean soup ranking is.

Which SQL query will *not* tell them what they want?

A. `SELECT SUM(ranking)/COUNT(ranking) FROM soupranking;`

B. `SELECT AVG(ranking) FROM soupranking;`

C. `SELECT (1.0*SUM(ranking))/COUNT(ranking) FROM soupranking;`

D. `SELECT SUM(ranking)/(1.0*COUNT(ranking)) FROM soupranking;`

sql内置小函数的考察，本质其实是这种计算要得出正确答案只能用函数or转化浮点数

如何转化浮点数？ → 分子 or 分母 * 1.0

Q13. The `PRIMARY KEY` constraint implies which other constraints (if any)?

A. `NOT NULL, UNIQUE`

B. `NOT NULL`

C. `UNIQUE`

D. No other constraints

主键约束。 对应的非主键约束时没有约束。