Week1

DML: Data Management Language

- Select, insert, update, delete

DDL: Data Definition Language

- Create, Alter, Drop

DCL: Data Control Language

- Grant, Revoke

TCL：控制事务。

COMMIT、ROLLBACK、SAVEPOINT

Logical data independence:

protection of the applications from changes in the logical structure of the data

Physical data independence:

protection of the conceptual schema (and applications) from physical layout changes

Week2

- 需要能够甄别出 ERD 中的错误
- 能根据 plain text 画出对应的 ERD，重点关注 Constraint 部分
- 箭头从 Entity 指向 Relationship

Entity

Rectangle

Weak Entity (1:N relationship; Strong Entity 和 relation 之间不能有箭头，否则破坏了 1:N 关系; Weak Entity 和 relation 之间必须为粗箭头)

Double Diamond – weak relationship

Double Rectangle – Weak Entity

Dotted Underline – Discriminator (Partial Key), weak entity must use discriminator and strong entity's PK to Form a Composite PK

Attributes

Eclipse

Multi-value Attributes

Double Eclipse

Composite Attributes

多画一条线连起来

Derived Attribute

Dot Eclipse

Primary Key

Underline

Foreign Key

Does not show

Partial Key

Doted Underline

Relationship (不需要画 Key，因为用连接的 Entity PKs 作为组合主键)

Diamond

Aggregation Relationship

Dotted Box

Conveys a relationship between two relationships

Key Constraint (N-to-1)

Thin Arrow

At most one

Total/Partial Participation (N-to-M)

Total: Thick Line; at least one

Partial: Thin Line

Combine Key and Total Participation

Thick Arrow

Exactly One

Cardinality

用 label 在线旁边表示

* 表示无限大

0···1 用细线

1...3 用粗线

IsA

这里没有 thick arrow（即一个父类必须属于一个子类），但是等同于 thick line + Disjoint

没有 cardinality

Overlap Constraint

Disjoint: 只能属于最多一种子类

标注在线旁边

Overlapping (default): 可以属于多个子类

无需表示

Covering Constraint

Total: 父类必须属于子类

Thick Line

Partial (default): 父类可以不属于子类

Thin Line

Week3

- 知道 RM 如何定义 FK constraint 才能满足 ERD，比如 FK 加上 NOT NULL 就能表示 exactly one

Real word Relation — RDBMS

- allow duplicated rows
- support ordering tuples and attributes
- allows “null”

Advantage/Disadvantage of NULL

Advantage:

- NULL can be useful because using an ordinary value with a specific meaning may not always work. 比如求 mean 的时候，如果我们用-1 代表 null 就会出错

Disadvantage:

- NULL may cause complications in the definition of many operations

CHAR VS VARCHAR

CHAR 是固定长度

VARCHAR 是动态长度

CREATE TABLE NAME(…);

DROP TABLE NAME CASCADE

ALTER TABLE

- **添加列**: ALTER TABLE Flight ADD test1 INTEGER, ADD test2 INTEGER; 不能同时添加 constraint
- **删除列**: ALTER TABLE Flight DROP test1;
- **添加 Constraint**: 只能先删除旧的外键约束，再添加一个新的外键约束
  - ALTER TABLE Orders ADD FOREIGN KEY (customer_id) REFERENCES Customers(customer_id);
  - ALTER TABLE Orders ADD UNIQUE (column_name);
- 删除 Constraint: **除了 NOT NULL 以外，别的都需要对应的 Constraint 名字来删除**
  - ALTER TABLE table_name DROP CONSTRAINT constraint_name;
  - ALTER TABLE table_name ALTER COLUMN column_name DROP NOT NULL;
- **重命名列**: ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name;
- **重命名表**: ALTER TABLE old_table_name RENAME TO new_table_name;
- 修改 Default Value 或者 data type
  - ALTER TABLE table_name ALTER COLUMN column_name SET DEFAULT 7.77;
  - ALTER TABLE table_name ALTER COLUMN column_name TYPE new_data_type;

PRIMARY KEY（主键）

- **唯一性**：每个表只能有一个主键（At most one per table），并且主键字段的值必须是唯一的。
- **不允许 NULL 值**：主键字段不能包含空值（Automatically disallows NULL values）
- **Composite PK 可以自己设定**，但必须满足唯一性约束。

CANDIDATE KEY

- **唯一性**：候选键中的字段也必须是唯一的（all must be declared as UNIQUE）。
- **最小性（Minimality）**：没有多余的属性，可以删除任何一个属性而不再满足唯一性的条件。这里的最小是不是指 size，而是说不能去掉任何属性。
- **可以包含 NULL 值**

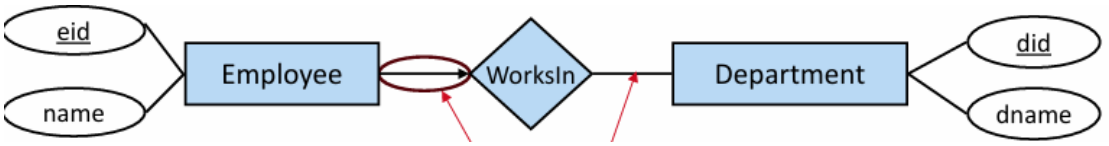
SUPER KEY

- **如果只满足唯一性，那么就叫做 Super Key**
- **所有 Candidate key 和 PK 都是 Super key**

Mapping relationship types From ERD To RM

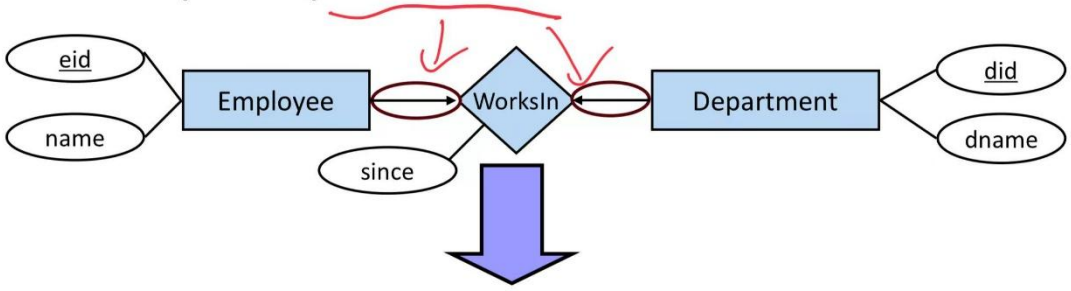
- 不包含 derived variable
- FK 的箭头应该指向被引用的主键
- 实线表示 PK，虚线表示 FK
- 可以有多条实线组成 Composite PK
- 一个 attribute 可以同时有虚线和实线，表示它同时属于 composite key 和 foreign key，通常出现在 Weak Entity 和 IsA relationship 中
- 在 IsA relationship 中，子类应该和父类共享 Primary Key

TUT 3

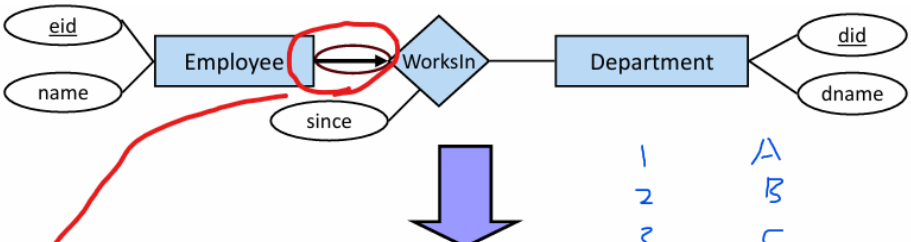


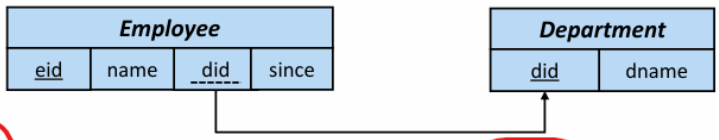
Employee works in at most 1 Department
1 Department can have 0 to Many Employees

› **Relationship with Key Constraints on *both sides***

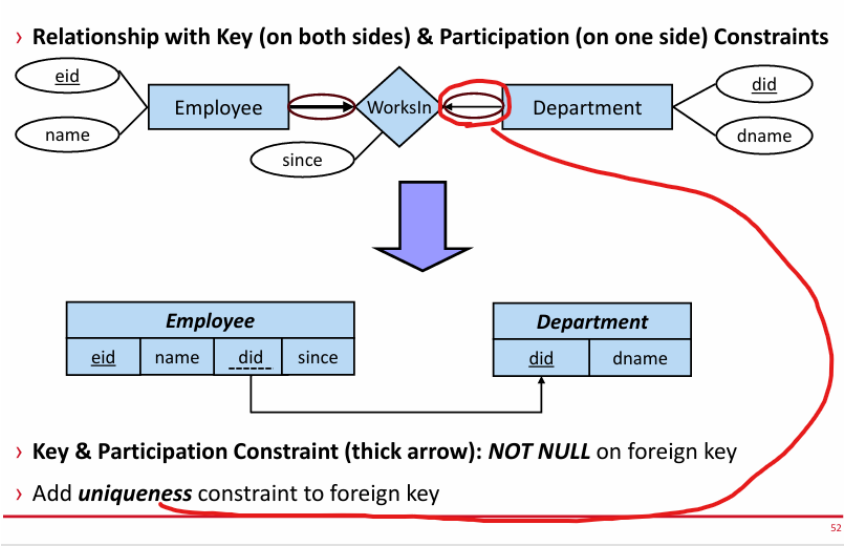


Add *uniqueness constraint* to foreign key did
Each Employee works in at most one Department
Each Department has at most one Employee





› **Key & Participation Constraint (thick arrow): *NOT NULL*** on foreign key



接下来看看哪些 INSERTION ROW 即使“违反”Constraint 也能插入成功

给出 Schema

```
CREATE TABLE Plane (
        plane_id VARCHAR(8) PRIMARY KEY,
        category VARCHAR(10) NOT NULL
                CHECK( category IN ('jet', 'turboprop')),
        capacity INTEGER NOT NULL
);
```

下面的需要注意

```
-- capacity should be integer (but still works):
INSERT INTO Plane VALUES ('ax-9999', 'turboprop', '50');
```

当你向一个 INTEGER 类型字段插入 '50' 这样的字符串文字时，它会自动尝试将其转换为整数

```
-- plane_id should be string (but still works):
INSERT INTO Plane VALUES (123456, 'jet', 255);
```

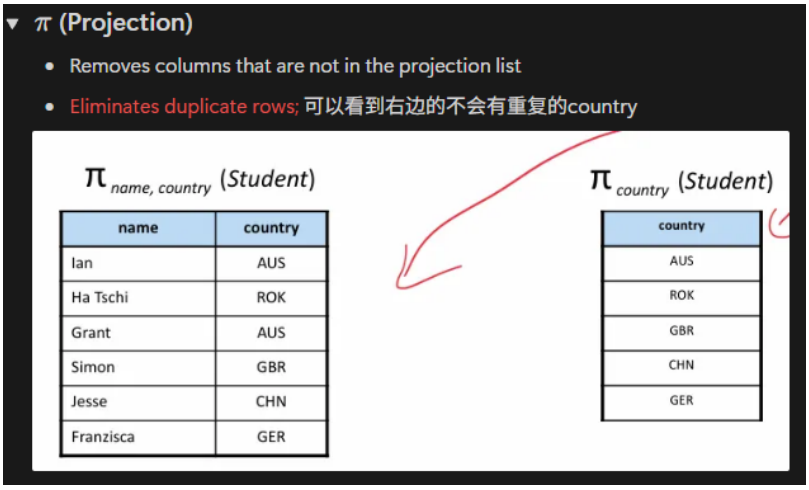
PostgreSQL 允许将 INTEGER 自动转换为 TEXT / VARCHAR

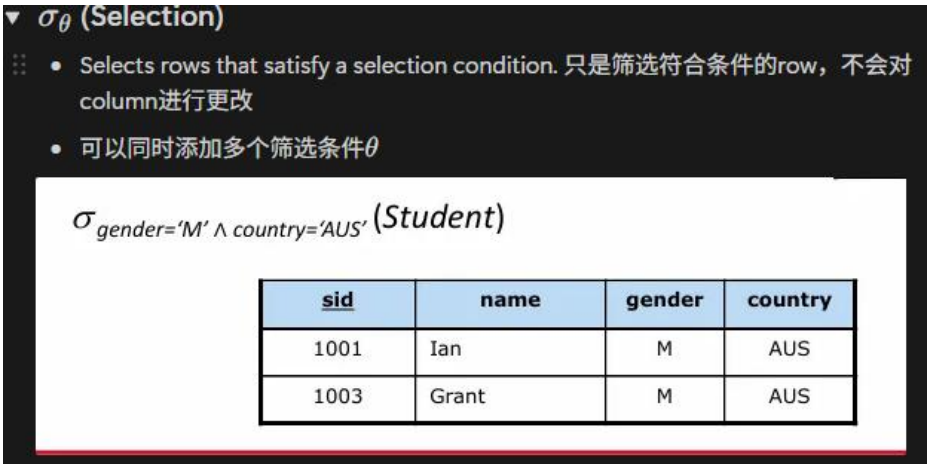
Week4

- 知道如何用 RA 表示一个 query
- 熟悉 plain description 对应的 set operation

RA (Relational Algebra)

- **6 Basic Operator: Projection (不会返回重复行), Selection, Cross, Union, Difference, Rename**





×(Cartesian/Cross product)

R	
A	B
α	1
β	2

X

S		
C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

=

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $R \times S = \{ts \mid t \in R \wedge s \in S\}$
- If two table have the same attribute name, then use the `rename` operation
- If R or S is empty, then $R \times S$ is also empty

∪ (Union)

Set Union $R \cup S$

- Definition: $R \cup S = \{t \mid t \in R \vee t \in S\}$

- R and S must have compatible schema, even for value domain
  - 下面只有在筛选出Student里面的sid列后才能进行difference计算，否则不满足上面的描述
  - 对于union和intersection也是同理

Student			
sid	name	gender	country
1001	Ian	M	AUS
1002	Ha Tschi	F	ROK
1003	Grant	M	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Franzisca	F	GER

their schmon is different

Student – Postgraduate ?

Postgraduate
sid
1003
1004
1005

Now they have same attributes

π_{sid}(Student) - Postgraduate

sol difference

sid
1001
1002
1006

— (Difference)

Set Difference $R - S$

- Definition: $R - S = \{ t \mid t \in R \wedge t \notin S \}$

- R and S must have compatible schema, even for value domain
  - 下面只有在筛选出Student里面的sid列后才能进行difference计算，否则不满足上面的描述
  - 对于union和intersection也是同理

Student

sid	name	gender	country
1001	Ian	M	AUS
1002	Ha Tschi	F	ROK
1003	Grant	M	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Franzisca	F	GER

their schmon is different

Student – Postgraduate ?

Postgraduate
sid
1003
1004
1005

Now they have same attributes

π_{sid}(Student) - Postgraduate

set difference

sid
1001
1002
1006

ρ (Rename)

只更改表名

$\rho_x(E)$: change relationship (i.e. 表名) E to x

既更改表名，又更改列名

$\rho_{x(A_1,A_2,\ldots)}(E)$: change relationship (i.e. 表名) E to x, also change the attributes name

比如下面的就是把"UnitOfStudy"换成了"UOS"，并且还重新命名了fields

$\rho_{UOS(ucode,title,credits)}(UnitOfStudy)$

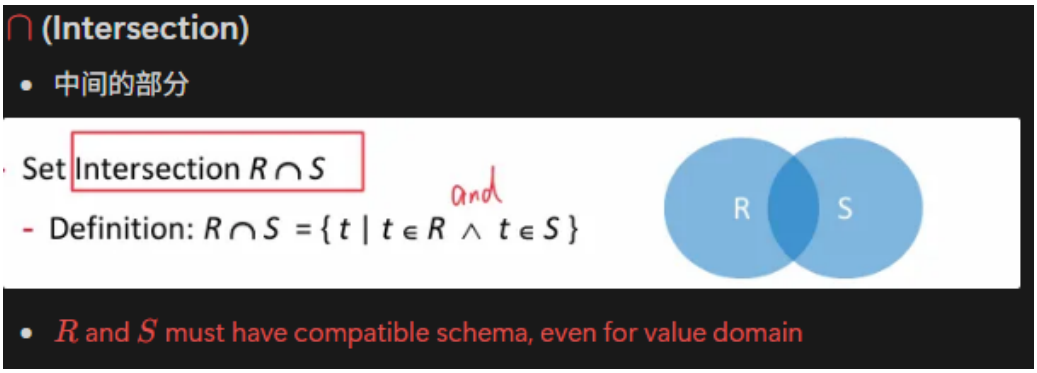
UnitOfStudy

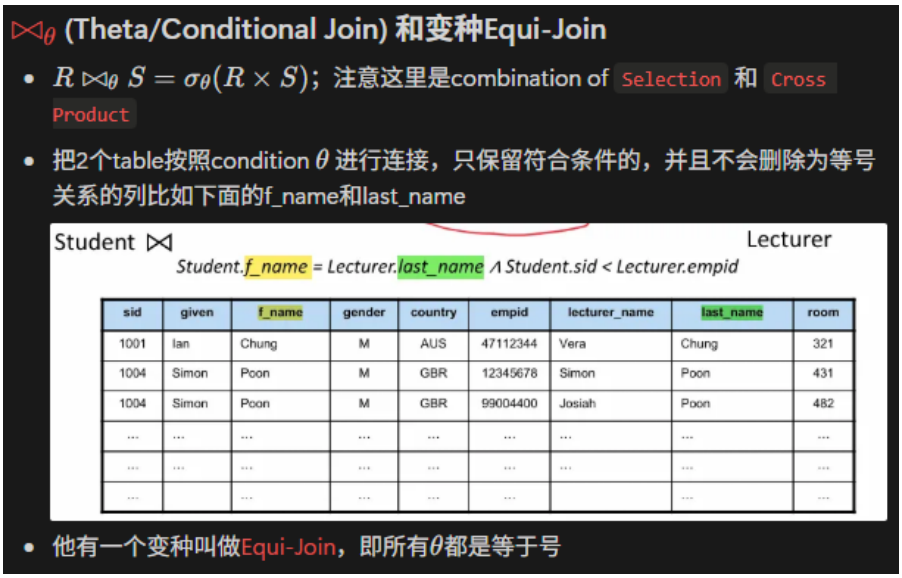
uos_code	title	points
COMPS138	Relational DBMS	6
COMPS318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMPS702	Thesis	18

UOS

ucode	title	credits
COMPS138	Relational DBMS	6
COMPS318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMPS702	Thesis	18

- 3 Derived: Intersection, Conditional Join, Natural Join





JOIN

- Implicit join is Cartesian join/CROSS join
  - 类似于 FROM A, B
- Inner join，特殊写法 using 替代 on; 省略 Inner
  - Theta Join（θ-Join）
  - Equi-Join 等值连接

- Natural Join
  - Natural Join Caveat

Natural Join Caveat：如果两个表中没有相同的列名，则 `NATURAL JOIN` 会返回笛卡尔积（不推荐）。

- $R \bowtie S = \pi_{unique\_attributes}(\sigma_{equality\_of\_common\_attributes}(R \times S))$

- Outer join
  - LEFF JOIN
    - NATURAL LEFT OUTER JOIN: 不需要 ON 语句，它会自动匹配两个表中相同名称的列，并以左表（Employee）为主表. 结果可能和 LEFT JOIN 不同，因为 NATURAL JOIN 可能匹配多个同名列，而 LEFT JOIN 只会匹配 ON 指定的列。
  - RIGHT JOIN
  - FULL JOIN: 不会返回重复的行。在右表没有孤儿行的情况下等同于 LEFT JOIN
- Natural Join vs. Explicit Equi-Join: 如果用 explicit equi-join，会导致连接字段出现 2 次,除非用 USING 替代 ON

Set operation UNION, INTERSECT, EXCEPT

- UNION 和 UNION ALL 的区别: UNION 会自动删除 duplicate，而 UNION ALL 会保留 duplicates
- 计算指定元素在两个 table 进行 XXX ALL operation 后的出现次数

Suppose a tuple occurs m times in R and n times in S, then it occurs:

- R UNION ALL S: $m + n$ times
- R INTERSECT ALL S: $min(m, n)$
- R EXCEPT ALL S: $max(0, m - n)$

- 两个 Set 需要有相同的的 Schema
  - Same number of columns: R and S must have the same number of columns.
  - Same data types: The corresponding columns in both relations must have compatible (typically the same) data types.
- EXCEPT 表示在 A 表不在 B 表
- 在使用 set operator 后可以使用 ORDER BY

TUT 4

- Additional (derived) operations:
  - E.g.: *intersection*, *join*. [Not essential, but very useful]
  - Equivalence: Intersection: $R \cap S = R - (R-S)$

    Join: $R \bowtie_\Theta S = \sigma_\Theta (R \times S)$

Intersection of
R = {1, 2, 3}
S = {3, 4} = {3}

Difference

R – S = {1, 2 }
R – (R - S) = {3}

$(1,2,3) - (1,2) = \{3\}$

- Example: List the names of all students enrolled in 'Relational DBMS'
  - One way is
    - π_{name} ($\sigma_{title='Relational\ DBMS'}$ ((Student ⋈ Enrolled) ⋈ UnitOfStudy))
  - Another (*more efficient*) way is:
    - π_{name} (Student ⋈ (Enrolled ⋈ ($\sigma_{title='Relational\ DBMS'}$ (UnitOfStudy))))

先 filter 几条

先 filter 一条

Student

sid	name	gender	country
1001	Ian	M	AUS
1002	Ha Tschi	F	ROK
1003	Grant	M	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Franzisca	F	GER

Enrolled

sid	uos_code	semester
1001	COMP5138	2023-S2
1002	COMP5702	2023-S2
1003	COMP5138	2023-S2
1006	COMP5318	2023-S2
1001	INFO6007	2023-S1
1003	ISYS3207	2023-S2

UnitOfStudy

uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	Thesis	18

```
SELECT S.Name
FROM Student S, Transcript T
WHERE S.studId = T.studId
AND T.uosCode IN ('INFO2005','INFO2120')
```

可以用 OR 在语义上替换 IN

```
Without using IN operator or a set construct:

SELECT S.name
FROM Student S, Transcript T
WHERE S.studId = T.studId
AND ( T.uosCode = 'INFO2005' OR T.uosCode = 'INFO2120' )
```

$$\pi_{Name}(\sigma_{uosCode='INFO2005' \vee uosCode='INFO2120'}(Student \bowtie Transcript))$$

Book (<u>isbn</u>, title, publisher, publicationYear)
Author (<u>aname</u>, birthdate)
Publisher (<u>pname</u>, address)
Wrote (<u>isbn</u>, <u>aname</u>) // which author wrote which book

c) $\pi_{aname}(\sigma_{title='A\ First\ Course\ in\ Datbase\ Systems'}(Book \bowtie Wrote))$ Natural join

List the author(s) of the book 'A First Course in Database Systems'.

d) $\pi_{address}(\sigma_{title='Databases' \vee title='Data\ Management'}(Publisher \bowtie_{pname=publisher} Book))$ theta join

Show the address of the publisher(s) of books titled 'Databases' or 'Data Management'.

c) Find all authors (name) who published at least one book with Acme Publishers

$$\pi_{aname}(\sigma_{publisher='Acme'}(Book \bowtie Wrote))$$

d) Find all authors (name) who never published a book with Acme Publishers.

$$\pi_{aname}(Author) - \pi_{aname}(\sigma_{publisher='Acme'}(Book \bowtie Wrote))$$

Most efficient answer

不会返回重复 row

Week5

- CHECK 是在插入数据前进行的，不管用不用 INITIALLY IMMEDIATE 等语句；
- CHECK 是针对行的
- CHECK 里面不支持 subquery，不支持聚合函数
- 需要注意检查的顺序
- **Only Domain constraints are modifiable**

Domain Constraints（static）

- NOT NULL / NULL
- DEFAULT
- CHECK -- User defined domain
    - 确保每个属性的值符合预定义的数据类型和取值规则。比如，成绩”只能是 ’A’，’B’，’C’，’D’，’F’ 之一
    - CREATE DOMAIN，用于创造新的数据类型
        - CREATE DOMAIN GradeDomain CHAR(1) DEFAULT 'P' CHECK (VALUE IN ('F', 'P', 'C', 'D', 'H'));

Key Constraints & Referential Integrity（static）

- Primary key
- Unique
- FOREIGN key
    - **Referential Integrity**
        - 主要关注的是 REFERENCES，后面这些 action 是额外的，因为默认是不会有任何操作的。
        - CASCADE
        - SET NULL
        - SET DEFAULT，如果对应的 col 没有 default constraint 会报错
        - ON DELETE/UPDATE CASCADE / SET NULL / SET DEFAULT
        - 两个可以同时存在，比如 ON DELETE .... ON UPDATE ....
    - **Delay**
        - NOT DEFERRABLE (默认的)，表明检测方式不能修改
        - DEFERRABLE INITIALLY IMMEDIATE，立刻检测
            - FOREIGN KEY (lecturer_id) REFERENCES Lecturer(lecturer_id) DEFERRABLE INITIALLY IMMEDIATE
        - DEFERRABLE INITIALLY DEFERRED，等 transaction 结束再检查
        - 修改上面 2 种的语法
            - SET CONSTRAINTS name IMMEDIATE
            - SET CONSTRAINTS name DEFERRED

Semantic Integrity Constraints（static）

- 一般用 ASSERTION 来解决跨表检测，因为 CHECK 无法跨表检测
- ASSERTION Cannot modify data
- 2 种语法
    - CREATE ASSERTION assertion-name CHECK (condition)

        - CREATE ASSERTION smallclub CHECK ((SELECT COUNT(*) FROM Sailors) + (SELECT COUNT(*) FROM Boats) < 10);
    - 通过 CREATE OR REPLACE FUNCTION 的形式来达成；不会 return new

```
--For a sailing club to be categorized as small, we require that the sum of the number of boats and
--number of sailors, be less than 10 at all times.

create or replace function maxcountsb() returns boolean as $maxc$
begin
if ((SELECT COUNT(sailors.sid) FROM Sailors)
        + (SELECT COUNT(boats.bid) FROM boats) < 9)
    then return true;
    else
    return false;
    end if;
end;
$maxc$ language plpgsql;

CREATE TABLE Boats (
            bid    INTEGER,
            PRIMARY KEY (bid),
            color  CHAR(10)--,
            --CHECK (maxcountsb())
            );
```

Trigger (Dynamic)

- 3 components
    - ON event: what activates the trigger
    - IF condition: test the condition's truth to determine whether to execute an action；不是所有的 trigger 都有 condition
    - THEN action: what happens if the condition is true
- Trigger can modify data
- 总共有 2 种 granularity
    - Row-level：每有一行进行改动，则检查一次
    - Statement-level：每条语句检查一次；不是每个事务，因此事务中如果出现多个语句，则这里会触发多次

语句级触发器

SQL Copy Caption

```sql
CREATE FUNCTION function_name()
RETURNS trigger AS $$
BEGIN
  -- 触发时执行的操作
  RETURN NULL; -- 对于 AFTER 触发器可以返回 NULL，表示不进行操作
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_name
AFTER INSERT OR UPDATE OR DELETE ON table_name -- 不能在一个语句中同
FOR EACH STATEMENT
EXECUTE FUNCTION function_name();
```

行级触发器

SQL Copy Caption

```sql
CREATE FUNCTION function_name()
RETURNS trigger AS $$
BEGIN
  -- 可以访问 NEW.column_name 或 OLD.column_name
  RETURN NEW;  -- BEFORE 触发器必须返回 NEW
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_name
BEFORE INSERT OR UPDATE OR DELETE ON table_name
FOR EACH ROW
EXECUTE FUNCTION function_name();
```

只能要么是 BEFORE 要么是 AFTER

其中针对 BEFORE, AFTER, INSERT, UPDATE, DELETE 都是可选项

- Use BEFORE triggers: Usually for checking integrity constraints
- Use AFTER triggers: Usually for integrity maintenance and update propagation

```sql
-- 定义一个触发函数：把对视图的 INSERT 转换为对 employee 表的 INSERT
CREATE FUNCTION insert_employee_into_view()
RETURNS trigger AS $$
BEGIN
  -- 把插入视图的行为改为插入真实表
  INSERT INTO employee(name, dept_id)
  VALUES (NEW.name, (SELECT id FROM department WHERE dept_name = NE
  RETURN NULL;
END;
$$ LANGUAGE plpgsql;

-- 定义 INSTEAD OF 触发器
CREATE TRIGGER employee_view_insert
INSTEAD OF INSERT ON employee_view
FOR EACH ROW
EXECUTE FUNCTION insert_employee_into_view();
```

这里是针对 view 的

TUT5

```
ALTER TABLE Requires ADD CONSTRAINT Requires_uoSCode_fk_UOS FOREIGN KEY
(uoSCode) REFERENCES UnitOfStudy(uoSCode) ON DELETE CASCADE;
```

ASSERTION

1. The number of students enrolled in a unit-of-study must equal the current enrolment;

```
CREATE ASSERTION EnrollmentAssert CHECK (
    NOT EXISTS (
        SELECT 1
        FROM UoSOffering o
        WHERE enrollment != (SELECT COUNT(*)
            FROM Transcript t
            WHERE t.uosCode=o.uosCode AND
            t.semester=o.semester AND
            t.year=o.year)
    )
);
```

2. The room assigned to a unit-of-study must have at least as many seats as the maximum allowed enrolment for the unit;

```
CREATE ASSERTION RoomCapacityAssert CHECK (
    NOT EXISTS (
        SELECT 1
        FROM (UoSOffering NATURAL JOIN Lecture)
            NATURAL JOIN ClassRoom
        WHERE seats < maxEnrollment
    )
);
```

TRIGGER

什么时候用 `RETURN NEW` 或 `RETURN NULL`？

触发器类型	推荐返回值	含义/说明
`BEFORE INSERT/UPDATE`	`RETURN NEW` 或 `RETURN NULL`	可以修改或取消即将执行的插入/更新操作。`NULL` 表示取消操作。
`AFTER INSERT/UPDATE/DELETE`	`RETURN NEW`（或 `RETURN OLD`）	必须返回一个记录，但返回值不会被使用，只是语法要求。

1. We can write a trigger to update the enrolment number for a unit of study offering *when a student is added* (in Transcript), as shown below:

```
CREATE OR REPLACE FUNCTION updateEnrol() RETURNS trigger AS $$
BEGIN
    UPDATE UoSOffering U
    SET enrollment = enrollment+1
    WHERE U.uosCOde = NEW.uosCode AND semester = NEW.semester
        AND year = NEW.year;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER UpdateEnrol AFTER INSERT ON Transcript
FOR EACH ROW EXECUTE PROCEDURE updateEnrol();
```

Week6

Set Comparison Operators/Nested Query

- [NOT] IN
- [NOT] EXISTS

```sql
-- 查询所有选修任何一门课程的学生
SELECT name
FROM Student s
WHERE EXISTS (
    SELECT 1
    FROM Enrolled e
    WHERE e.student_id = s.id
);
```

- ALL

```sql
-- 查询比所有 "COMP5138" 选课学生的 GPA 都高的学生
SELECT name
FROM Student
WHERE gpa > ALL (
    SELECT s2.gpa
    FROM Student s2
    JOIN Enrolled e2 ON s2.id = e2.student_id
    WHERE e2.uos_code = 'COMP5138'
);
```

- SOME

```sql
-- 查询 GPA 高于部分（至少一个） "COMP5138" 学生的学生
SELECT name
FROM Student
WHERE gpa > SOME (
    SELECT s2.gpa
    FROM Student s2
    JOIN Enrolled e2 ON s2.id = e2.student_id
    WHERE e2.uos_code = 'COMP5138'
);
```

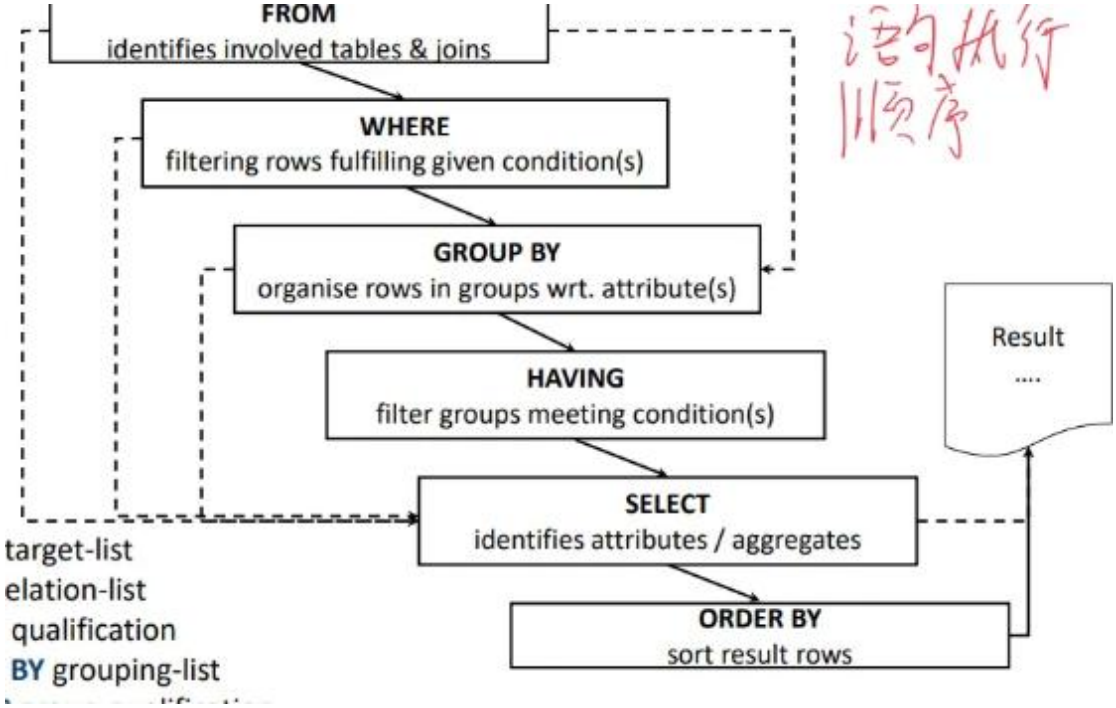
View

- 不是真实存在的
- CREATE VIEW view_name AS ⋯ ;
- CREATE VIEW student_enrollment AS SELECT sid, name, title, semester FROM student NATURAL JOIN Enrolled NATURAL JOIN unitofstudy;

Aggregate function

- COUNT(*)计算 NULL，别的都不算，包括 COUNT(x)
- 不能 MIN(AVG())
- 都对 DUPLICATE 起效，除非 COUNT(DISTINCT x)
- If you use GROUP BY function, then in SELECT or HAVING line, you can only include aggregate function or the attribute you use for GROUP BY
- Predicates in the HAVING clause are applied after the formation of groups, whereas predicates in the WHERE clause are applied before forming groups

SQL 语句执行顺序



NULL and Three-valued Logic

- 任何和 NULL 进行的运算都是 NULL，比如 5+NULL=NULL
- Any comparison with NULL returns unknown，比如 5<NULL 是 unkown
    - Result of WHERE clause predicate is treated as false if it evaluates to unknown
    - SELECT sid FROM enrolled WHERE marks < 50;
        - ignores all students without a mark, that is if a student with null mark, then we will not out put it in above query
- Three-Valued Logic
    - UNKOWN OR TRUE = True
    - UNKOWN AND FALSE = False
    - 别的都是 UNKOWN

Week6

a) a = 10

Answer: All tuples with a being 10 combined with any value for b, including NULL. Examples: (10, 0), (10, 1), ..., (10,-1), ..., (10, NULL)

d) a < 10 AND NOT b = 20

Answer: Similar to the previous answer: All tuples where a < 10 and not NULL, and b ≠ 20 and also not NULL.

可以看到 where 会把带有 NULL 的判断归类为 False

a) Which lecturers (by id and name) have taught both 'INFO2120' and 'INFO3404'? Write a SQL query to answer this question using a SET operator.

Answer :

```
SELECT id, name
FROM AcademicStaff JOIN UoSOffering ON id=instructorId
WHERE uosCode = 'INFO2120'
INTERSECT
SELECT id, name
FROM AcademicStaff JOIN UoSOffering ON id=instructorId
WHERE uosCode = 'INFO3404';
```

b) Which lecturers (by id and name) have taught both 'INFO2120' and 'INFO3404'? Answer this using a sub-query without SET operators. Make sure your result doesn't include duplicates.

Answer :

```
SELECT DISTINCT id, name
FROM AcademicStaff JOIN UoSOffering ON id=instructorId
WHERE uosCode = 'INFO2120'
AND id IN ( SELECT instructorId
 FROM UoSOffering
 WHERE uosCode = 'INFO3404' );
```

c) Write a SQL query to give the **student IDs** of all students who have enrolled in only one lecture using GROUP BY, and order the result by student ID. A lecture is a unit_of_study in a semester of a year.

Answer:

```
SELECT studId
FROM Transcript
GROUP BY studId
HAVING count(*) = 1
ORDER BY studId;
```

需要注意的是 HAVING 是对 GROUP 进一步进行 filter

g) [Advanced, Optional] Write a SQL query to give the **student IDs** of all students who have enrolled in only one **unit_of_study**, and order the result by student ID. Note that, a student can enrol in the same unit_of_study multiple times, which is still counted as one unit_of_study.

Answer :

```
SELECT studId
FROM Transcript
GROUP BY studId
HAVING count(DISTINCT uoSCode) = 1
ORDER BY studId;
```

h) [Advanced, Optional] Write a SQL query to give the **student IDs** and **names** of all students who have enrolled in only one unit_of_study, and order the result by student ID. Note that, a student can enrol in the same unit_of_study multiple times, which is still counted as one unit_of_study.

Answer :

```
SELECT studId, name
```

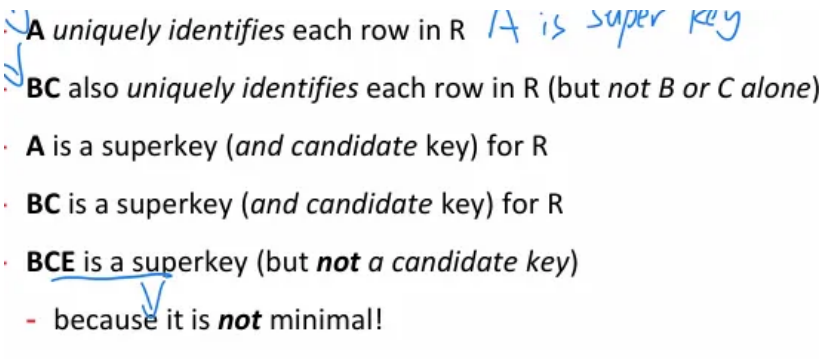
Select
name
studId
From Student
where
studId in (g)


```
FROM Student NATURAL JOIN (
 SELECT DISTINCT studId, uoSCode
 FROM Transcript ) AS T
GROUP BY studId, name
HAVING count(*) = 1
ORDER BY studId;
```

Week8

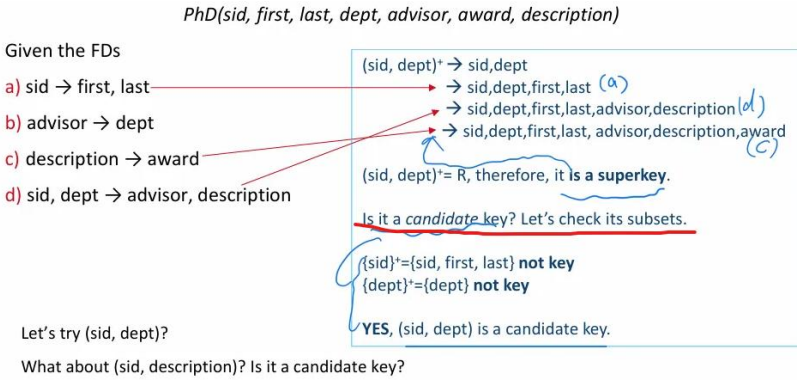
Functional Dependency X->Y

- It's an n-1 relationship, hence 1-1 relationship. Namely, **X 的值唯一确定 Y 的值**。即只要你知道 X，就能**唯一**确定 Y
- 我们可以通过查看一个实例来判断某个函数依赖是否不成立，即通过一个反例可以推翻 FD
- 无法通过查看任何数量的实例来确定 FD 是否成立，因为函数依赖描述的是关系中所有可能合法实例的规则。必须查看所有实例才行
- Inference Rules
  - **Reflexivity (Trivial FDs): A 属于 B 则，A->B**
  - **Augmentation: A->B 则 AC->BC**
  - **Transitivity: A->B, B->C, A->C**
  - **Decomposition: A->BC 则 A->B and A->C**
  - **Union: A->B, A->C 则 A->BC**



Closure of Attribute

- 如果一个 set of attributes 通过已知的 FD 可以得到所有的 relation 里面的 element，它就是 Super key
- 通过甄别只在 FD 左边出现的，但是没在 FD 右边出现的 attributes 可以得到 Candidate key



Schema Decomposition (2 methods)

- **Dependency preservation**: No FDs are lost in the decomposition

- Lossless-join (also called non-additive) decomposition: Re-joining (natural join) a decomposition of R should give us back R!

Normal Forms

- Higher restriction **Not necessarily good for mostly retrieval operations**";
- Higher NF include Lower NF
- 1NF: 确保 atomic
- 2NF: 消除 partial dependency
- 3NF: 消除 Transitive dependency
- BCNF: 确保 X 必须是超键
- 4NF: solve MVD (Multivalued Dependency)

1. 满足atomic — From 1FN
2. No partial Dependencies

   A partial dependency is a non-trivial functional dependency $X \rightarrow Y$ in relation R, where X is a strict (proper) subset of some key for R, and Y is not part of any key.
   也就是说，不能只有主键中的一部分functional determine另外一个非主键element
3. A relation schema R is in **Third Normal Form (3NF) if for every functional dependency $X \rightarrow Y \in F^+$, at least one of the following holds:**
   a. $X \rightarrow Y$ is a **trivial** functional dependency: $Y \subseteq X$
   b. X is a **superkey** for R
   c. $Y \subseteq$ some candidate key of R

   简而言之就是："every non-key attribute is not transitively functionally dependent on a key"；每个非key attribute不能间接依赖于key；这里是允许函数依赖的"箭头"不从超键出发（因为c）
4. 对于所有non-trivial $X \rightarrow Y$，X 是 R 的一个**superkey，即**所有函数依赖的"箭头"必须从超键出发
5. 对于所有形式为 $X \xrightarrow{\rightarrow} Y$ 的多值依赖，必须满足以下 **至少一个** 条件：
   a. trivial MVD：
      - $X \xrightarrow{\rightarrow} Y$ 是trivial MVD，即满足以下任一条件：
        - $Y \subseteq X$，即 Y 是 X 的子集，或
        - $X \cup Y = R$，即 X 和 Y 的并集包含关系 R 的所有属性。
   b. **超键条件：**
      - X 是关系 R 的**超键**

employee_name	project_id	personal_phone_number
Bob	P1	047012345
Bob	P3	046098765
Bob	P1	046098765
Bob	P3	047012345
Lily	P1	045067543
Fiona	P7	043085432

Is this relation in 4NF?

No: There is at least one non-trivial multivalued dependency

存在多值依赖：$employee\_name \twoheadrightarrow Project\_id$,

而Y不是X的子集，并且X UNION Y不是R

同时$employee_name$ is not a superkey.

TUT8

FD

Consider the following table called *PhD*:

SID	first	last	dept	advisor	award	description
1234	Brian	Cox	IT	Codd	Rejected	Work not deemed sufficient
3456	Albert	Einstein	IT	Boyce	Conditional	Accepted with minor corrections
3456	Albert	Einstein	Physics	Newton	Accepted	Accepted with no corrections
7546	Alan	Turing	IT	Codd	Accepted	Accepted with no corrections
4879	Brian	Cox	Physics	Newton	Conditional	Accepted with minor corrections
4879	Brian	Cox	Media	Attenborough	Accepted	Accepted with no corrections

What *functional dependencies* might exist in the **PhD relation** above?

Sid → name

Br.an Cox → sid is not

dept → advisor

award → description ✓

description → award ✓

branch_name	assests	city	loan_no	customer_name	amount
Mall St	9000000	Sydney	17	Jones	1000
Logan	5000000	Melbourne	23	Smith	2000
Queen	500000	Perth	15	Hayes	1500
Mall St	9000000	Sydney	14	Jackson	1500
King George	10000000	Brisbane	93	Carry	500
Queen	500000	Perth	25	Glenn	2500
Bondi	15000000	Adelaide	10	Brooks	2500
Logan	5000000	Melbourne	30	Johnson	750

› Do you see any other functional dependency?

- For instance, are the following FDs correct?

loan_no → customer_name, amount? **YES**

loan_no → branch_name? **YES**

city → customer_name? **NO**

city → assets? **YES**

sydney — Jackson, Jones

Value are unique

I. destination, departs, airline → gate

II. gate → airline

III. contact → name

IV. name, departs → gate, pickup

V. gate, departs → destination

b) Consider the following collection of tuples. Why is this instance not a legal state for the database?

Destination	Departs	Airline	Gate	Name	Contact	Pickup
Berlin	1/06/2012 11:25	Lufthansa	3	Justin Thyme	0416594563	1
Madrid	1/07/2012 14:30	Iberian	4	Willy Makit	0497699256	2
London	3/05/2012 6:10	British Airways	7	Hugo First	0433574387	5
Moscow	1/07/2012 17:50	Aeroflot	6	Rick OhChet	0416594563	7
Berlin	1/06/2012 11:25	Qantas	1	Dick Taite	0469254233	4
Kuala Lumpur	1/08/2012 14:30	Cathay	7	Hugo First	0433574387	2
Singapore	1/08/2012 14:30	Qantas	2	Hugo First	0433574387	2
London	1/07/2012 17:50	Lufthansa	3	Justin Thyme	0413456789	4

需要根据 FD 来判断，即一个 X 对应一个 Y，但是 Y 不需要不同

British Airways and Cathay can't share the same gate 7.

Rick can't use Justin's phone number (but it's fine that Justin has 2 numbers).

Hugo can't make two flights at the same time on 1/8/12. Maybe he changed flights and the old flight didn't get removed.

Attribute Closure

a) Is (contact, departs, airline) a candidate key from the above functional dependencies. Can you find an alternative?

这里就是根据 Closure of attributes 来判断 Candidate key；但是因为 contact 和 depart 只在 FD 的左边出现过，因此(contact, departs, airline)最多只能是 Super key，因为不满足 minimal。

b) Is the relation in 3NF?

gate->airline does NOT meet the 3NF restriction that either the LHS is a superkey or at least for the RHS to be part of a key. (hence cannot be in 3NF). In addition，还要检查 2NF 和 1NF, contact->name 不满足 2NF(Partial dependenct)，因为 contact 是

candidate key 中的一个；下面是老师给的例子

Teacher_name	UnitOfStudy	Teacher_position
Mary	COMP9120	Lecturer
Mary	COMP5313	Lecturer

In this, we assume that Teacher_name -> Teacher_position, and {Teacher_name, UnitOfStudy} -> Teacher_position. As such, although the relation doesn't have any transitive dependencies, it is not in 3NF because it doesn't satisfy 2NF.

Lossless-Join

c) Explain whether it is a lossless-join decomposition to decompose the relation into the following:

R1(destination, departs, gate)

R2(contact, departs, pickup)

R3(gate, airline)

R4(contact, name)

这里因为 R3 和 R4 是直接和本来的 FD 相关，因此可以从 R 中删除 airline, name 这 2 个 attributes，变成 R7(dest, dep, gate, contact, pickup)；这里之所以不删除 gate 和 contact 是因为 R3 和 R7 的 intersection 为 gate 才能保证 gate 是 R3 的 superkey。现在来看 R7 (dest, dep, gate, contact, pickup)，他如果分解成 R1 和 R2，则它们的 intersection 是(departs)，根据现有的 FDs，我们足以推断出 departs 的 closure 不能包含所有在 R1 或 R2 里面的 attributes，因此不是 lossless 的。正确的步骤是一个 Ri 一个 Ri 的检查，比如先从 R 里面分出 R3，再分出 R4，然后每一步检查分的是不是满足 lossless。

d) Give a lossless-join decomposition of the original relation into BCNF relations.

很重要!!!

首先不满足 BCNF 的有

所有 FD

那么，我们先拿最小的 FD 开刀，因此分出了（注意这里是一步步来的，这里为了方便所以一起分了）

R3(gate, airline),

R4(contact, name)

这里在 R3 和 R4 中，都满足 BCNF 因此不用再分；此外因为

gate → airline

contact → name

所以我们分别用 gate 和 contact 作为 R3 和 R4 的 key,

现在还剩下 R7(destination, departs, gate, contact, pickup)，还是不满足 BCNF，因为

contact, departs → gate, pickup；从(a)中知道(contact,depart)是 super key，所以没问题

gate, departs → destination；但是 gate 和 departs 不是 super key，所以有问题

因此需要把(gate, departs, destination)分成 R5，在这里面，我们需要把 gate 和 departs 作为 super key。

最后剩下 R6(departs, gate, contact, pickup)，而我们发现 R6 也满足 BCNF 因为

contact, departs → gate, pickup

最后得到作为 decoposed set

R3(gate, airline),

R4(contact, name)

R5(destination, departs, gate) R6(departs, gate, contact, pickup)

Week9

Transaction Syntax

这里会在 SQL 失败时回滚，如果没失败就不回滚；ABORT 和 ROLLBACK 等价

在 postgreSQL 里面，会自动确保 Atomic，因此即使没有 ROLLBACK，在 SQL 失败的时候也不会修改数据；但是仍然需要 ROLLBACK 来结束事务

```
BEGIN;
    SQL;
COMMIT;
ROLLBACK/ABORT;
```

ACID

Atomicity — log

- A real-world transaction is expected to happen or not happen at all
- 通过 ROLLBACK 达成

Consistency

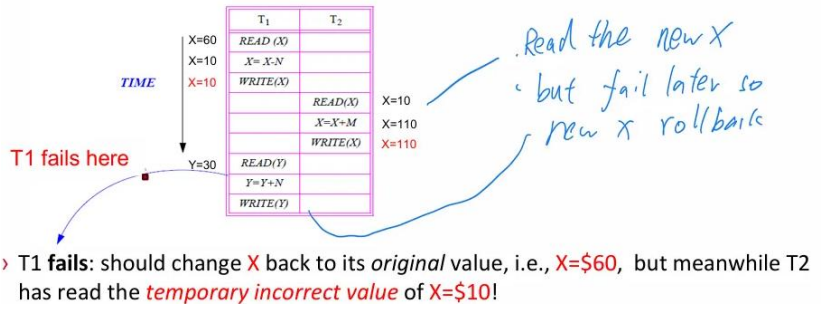
- 数据库**无法修正你写错的业务逻辑（Business logic）**。如果你的事务代码错了，数据库再强也没法保你一致。
- can only modify data in allowed ways. 即需要满足数据库中的 Constraints

Durability — log, Recover manager, exclusive lock

- Once a transaction is committed, its effects should persist in a database, and these effects should be permanent even if the system crashes.
- Recovery Manager of DBMS responsible for this part
- If a transaction aborts, depending on the recovery protocol, use the log to undo/redo the transaction.
    - Undo: 将数据项恢复到事务开始之前的值。
    - Redo: 将已经提交的事务操作再“做一遍”，确保它们的更改是永久性的。

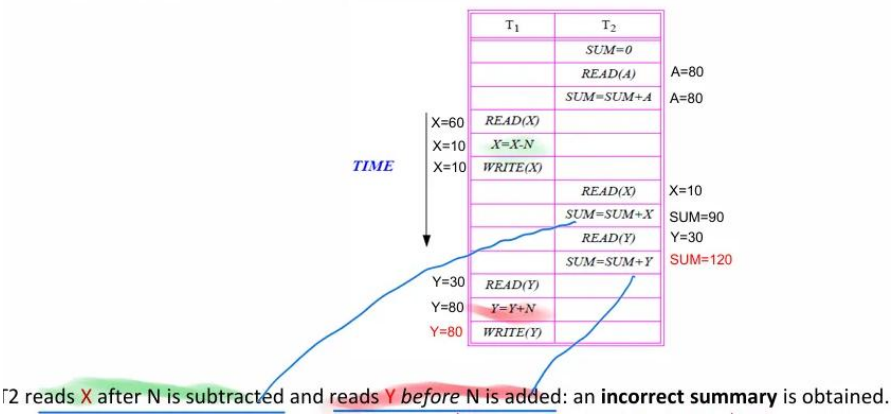
Isolation — Concurrently, interleaved

- Transactions can run concurrently; meaning their operations can be interleaved.
- **3 Concurrent Access Issues**
    - **Temporary update**: 一个事务对某项数据做了修改，**另一个事务读取了该临时值**，但前一个事务最后却失败或回滚，导致另一个事务**基于错误值作出错误决策**。

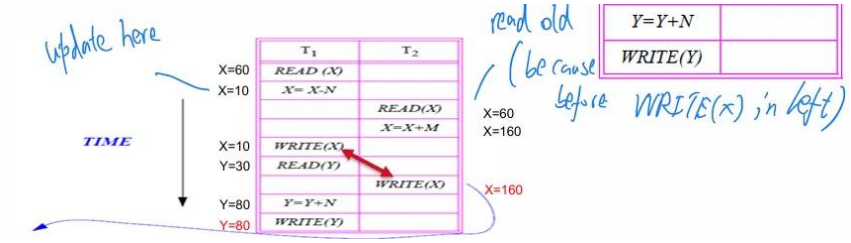


    - **Incorrect summary**: 一个事务对某些数据做聚合操作（如 SUM），但这些数据**在聚合过程中被其他事务修改了部分值**，导致聚合结果**混合**

了旧值和新值



- **Lost update**：两个事务几乎同时对同一数据项进行读取并修改，但**后一个事务的写操作覆盖了前一个事务的写操作**，导致前一个事务的修改被“**丢失**”。



- 4 Isolation Level

```
SET TRANSACTION ISOLATION LEVEL
{ SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }


-- Example
BEGIN;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- 这里开始写查询或更新语句
SELECT * FROM my_table;

COMMIT;
```

  - **Read Uncommitted** — Dirty Read
    - 允许事务读取其他尚未提交事务的数据，也叫做 Dirty Read。
    - PostgreSQL 不支持此隔离级别，直接转换为 Read Committed。
  - **Read Committed** — Solve **Temporary update**
    - 每次查询只能读取已经提交的数据。no dirty reads.
    - 防止脏读，但在同一个事务中两次读取同一行数据，可能得到不同的结果。
    - **是 PostgreSQL 的默认隔离级别**。
  - **Repeatable Read** — Solve **incorrect summary problems**
    - 只能“看到”在它开始之前已经提交的事务所做的更改。

        - transaction 看不到：
            - 其他事务还没有提交的数据 — 避免 Read Uncommitted
            - 以及它执行期间，其他事务新提交的更改 — 避免 Read Committed
    - **Serializable** — Solve **lost update problem**
        - 整个执行结果就像是所有事务串行执行一样。

Serializability (Optimistic)

- 你可以通过分析两个事务之间的调度是否满足 **serializability**（特别是 **conflict serializability**）来判断它们是否满足最高的隔离级别 SERIALIZABLE
- **Conflict Serializability -> Serializability**
- **Serializability**
    - A schedule is serializable if and only if it is equivalent to **some** serial schedule；some 表示有一个就行
    - Serializability is expensive to check
- **Conflict Serializability**
    - 如果你能**通过交换不冲突的操作**，把这个调度转换成一个 **串行调度（所有事务一个接一个执行）**，那么它是 conflict serializable; A schedule is conflict serializable if it is conflict equivalent to a serial schedule.
    - **Conflict Pair**
        - **Not a conflict pair (A1, A2)：**
            - A1 读 A，A2 读 A（Read-Read）→ 没有冲突（读操作不会影响另一个读）
            - A1 读 A，A2 读 B（不同数据项）→ 没有冲突
            - A1 写 A，A2 读 B（不同数据项）→ 没有冲突
        - **conflict pair (A1, A2)：**
            - A1 读 A，A2 写 A（Read-Write）→ 有冲突：T1 的读取可能受 T2 的写入影响，或反过来
            - A1 写 A，A2 读 A（Write-Read）→ 有冲突：顺序不同，读到的值不同
            - A1 写 A，A2 写 A（Write-Write）→ 有冲突：最终 A 的值取决于谁最后写
    - 通过 Non-conflicting swappings 或 Precedence Graph 来 check

Lock (Optimistic)

- **粒度大（粗粒度）**：并发性差，多个事务难以同时操作不同的数据项。 (too coarse) - no effective concurrency
- **粒度小（细粒度）**：虽然并发性高，但加锁和管理的**开销大**。(too fine) - lock overhead high

- Shared Lock (S) & Exclusive Lock(X); T1 持有，T2 申请

Read locks: "**Shared**" lock (S)
Write locks: "**Exclusive**" lock (X)

T2 Requests \ Held by T1	**Shared** Read	**Exclusive** Write
Shared Read	OK	T2 wait on T1
Exclusive Write	T2 wait on T1	T2 wait on T1

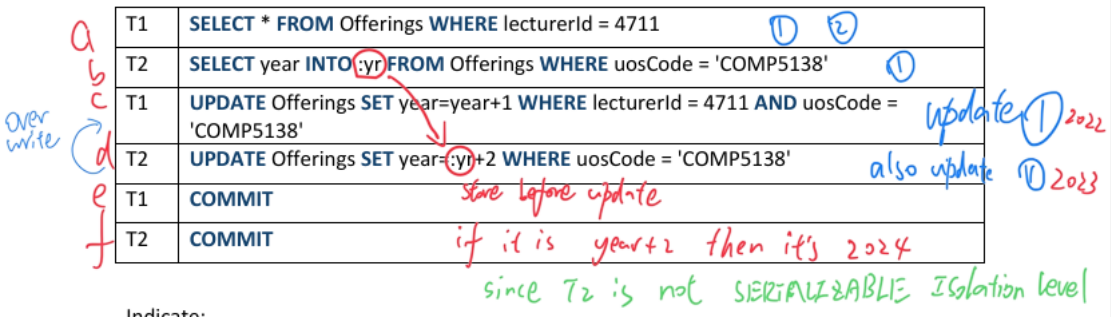
- **在使用完某个数据项后立即释放锁可能会导致起不到锁的作用!**
- **2PL**
  - 只能先加锁(growing)，再释放锁(Shrinking)
  - 在**没有死锁**和**没有事务失败**的前提下，2PL 能够确保调度的执行是**等价于某种串行执行顺序**的，即满足可串行性要求
  - Basic
    - 可以在 commit 前释放锁
  - Strict
    - 只能在 commit 后释放锁
- **Dead Lock**
  - Example
    - 事务 T1 已经**持有了数据 A 的锁**，并且**正在请求 B 的锁**；
    - 同时，事务 T2 **持有了 B 的锁**，并且**正在请求 A 的锁**；
    - 两个事务相互等待，谁都无法继续执行，这就形成了**死锁**。
  - Solution
    - Deadlock Prevention — Static 2PL
      - **每个事务**在开始时**预声明其读取集（共享锁）和写入集（排他锁）**。
    - Deadlock Detection — TIMEOUT

TUT9

uosCode	year	semester	lecturerId
COMP5138	2021	S1	4711
INFO2120	2021	S2	4711

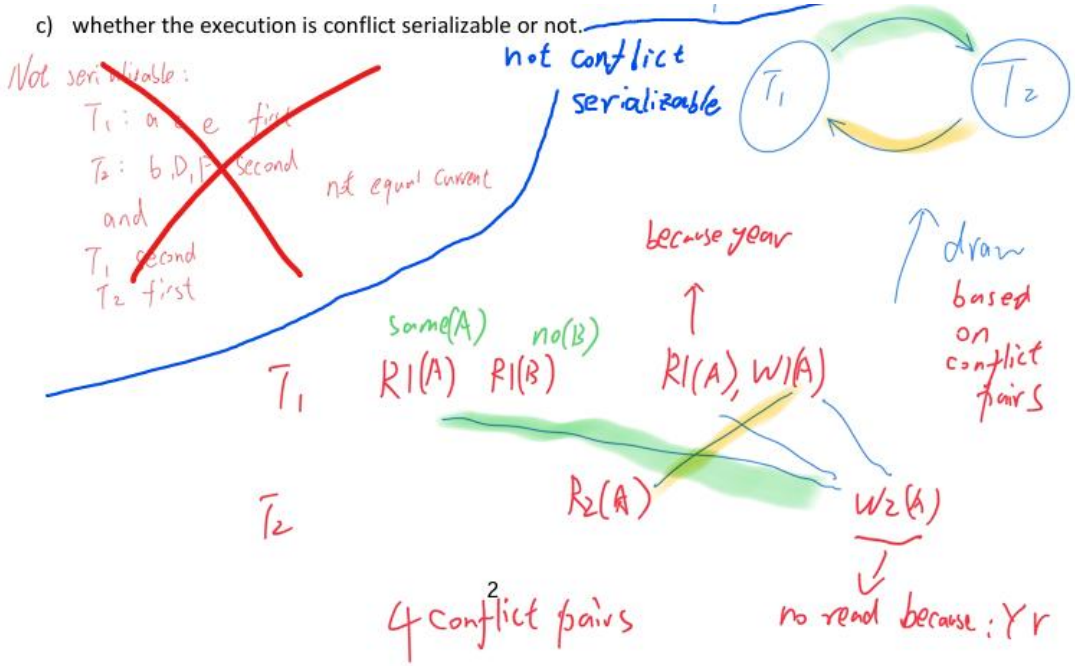
Consider the following hypothetical interleaved execution of two transactions T1 and T2 in a DBMS where concurrency control (such as locking) is not done; that is, each statement is executed as it is submitted, using the most up-to-date values of the database contents.

T1	**SELECT** * **FROM** Offerings **WHERE** lecturerId = 4711
T2	**SELECT** year **INTO** :yr **FROM** Offerings **WHERE** uosCode = 'COMP5138'
T1	**UPDATE** Offerings **SET** year=year+1 **WHERE** lecturerId = 4711 **AND** uosCode = 'COMP5138'
T2	**UPDATE** Offerings **SET** year=:yr+2 **WHERE** uosCode = 'COMP5138'
T1	**COMMIT**
T2	**COMMIT**



b) whether the execution produces any update anomalies Lost update

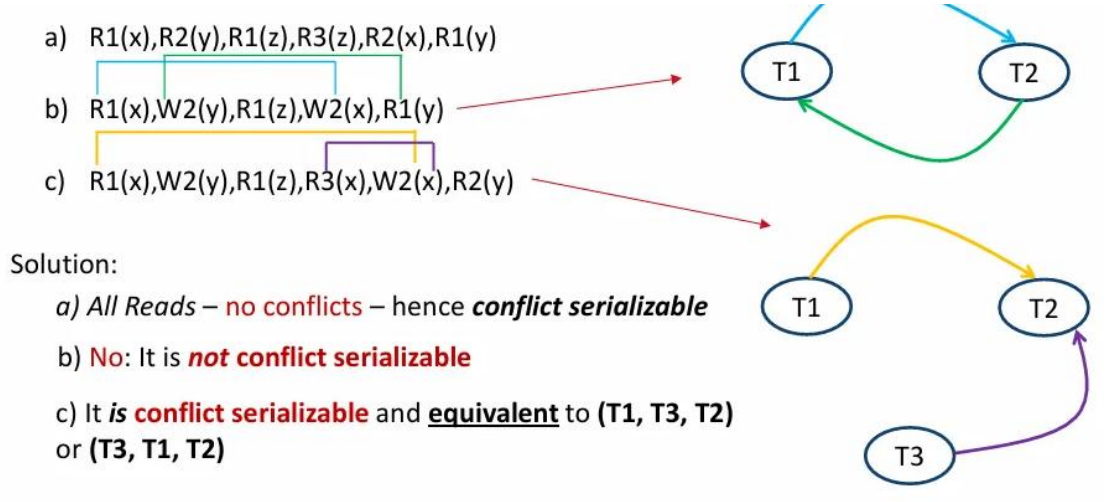
c) whether the execution is conflict serializable or not.



只用确定有 LOOP 就行，不需要识别出所有的 confilct pair

SELECT 只是 READ ROW

UPDATE，DELETE 是 READ+WRITE ROW



可以看到 conflict pair 对于 Reaction 的出现顺序有要求

T1 和 T3 都要在 T2 前面

Attention in SQL syntax

- 英文
  - chronologically 按时间顺序
  - ordinal ranking: 每个条目都有独立编号，即使有相同值，也不会并列；ROW_NUMBER()
  - dense ranking：如果有相同值，会并列排名，下一个排名不会跳过；DENSE_RANK()
  - Standard ranking（或叫 sparse ranking）：有相同值时会并列，下一个排名会跳过；RANK()
- REFERENCE KEY MUST EITHER BE CANDIDATE KEY OR UNIQUE
- 不是每个表都必须有主键，但是绝大部分正规设计的表都会有主键。
- BETWEEN .. AND... 是 inclusive 的，而且可以对 String 使用
- Int / Int 返回的还是 INT，并且是直接截断小数部位，不会四舍五入
- LIKE 和 LOWER 连用可以解决忽略大小写的问题
- NULL 用 IS 判断
- 需要注意的是，别名需要用双引号 或者 不带引号，不能用单引号
- 用 YEAR 作为别名的时候要用到 AS
- 关于 SELECT 中别名的引用
  - WHERE 里面不能用
  - HAVING 里面不能用
  - GROUP BY 可以
- natural join 不用关心表的顺序，它会自动找出其中相匹配的 col 进行连接，并且会合并同名列
- The set operators require that all set relations have the /same schema/.
- INFINITY 和 -INFINITY 和 NaN 都是 FLOAT 类型，并且 INFINITY = INFINITY
- TIME, DATE, TIMESTAMP
  - CURRENT_TIMESTAMP, CURRENT_DATE, CURRENT_TIME
  - 可以用 EXTRACT 提取对应的,比如 TIME 能提取 MINUTE，但是 DATE 不行
    - DAY
    - MONTH
    - YEAR
    - HOUR
    - MINUTE
    - SECOND
  - DOY (day-of-the-year)
  - TIMEZONE, TIMEZONE_HOUR, TIMEZONE_MINUTE
    - SELECT EXTRACT(**TIMEZONE** FROM TIMESTAMP '2025-05-04 12:00:00+08:00'); 像这里是 28800 秒 对应的是 8 小时
    - SELECT EXTRACT(TIMEZONE_**HOUR** FROM TIMESTAMP '2025-05-04 12:00:00-08:00'); -8 因为小时部分为-8
    - SELECT EXTRACT(TIMEZONE_**MINUTE** FROM TIMESTAMP '2025-05-04 12:00:00+08:00'); 0 因为分钟部分为 0

    - INTERVAL
        - INTERVAL '5 days'
    - DATE – DATE 得到的是相差的天数，为 INT
    - DATE – INTERVAL 得到的还是 DATE 类型
- Format String
    - TO_CHAR(5, 'FM00') 会返回 05，即使数字是单一数字，FM 也会强制将它变成两位数字。
    - TO_CHAR(5, 'FM00x') 会返回 05x，因为 x 并不会被识别成填充符号。
    - TO_CHAR(-5, 'S00') 会返回 -05
- CASE WHEN … THEN … ELSE … END

```
SELECT film_id, title, release_year, rating,
    CASE
        WHEN rating = 'G' THEN 'General Audiences. All Ages Admitted.'
        WHEN rating = 'PG' THEN 'Parental Guidance Suggested. Some Material May Not Be Suitable For Children.'
        WHEN rating = 'PG-13' THEN 'Parents Strongly Cautioned. Some Material May Be Inappropriate For Children u
        WHEN rating = 'R' THEN 'Restricted. Children Under 17 Require Accompanying Parent or Adult Guardian.'
        WHEN rating = 'NC-17' THEN 'No One 17 and Under Admitted.'
    END rating_def
FROM Film
WHERE release_year BETWEEN 2002 AND 2006
```

- COALESCE()

```
SELECT film_id, title, release_year, COALESCE(rating, 'TBD') rating
FROM Film
ORDER BY release_year DESC, title;
```

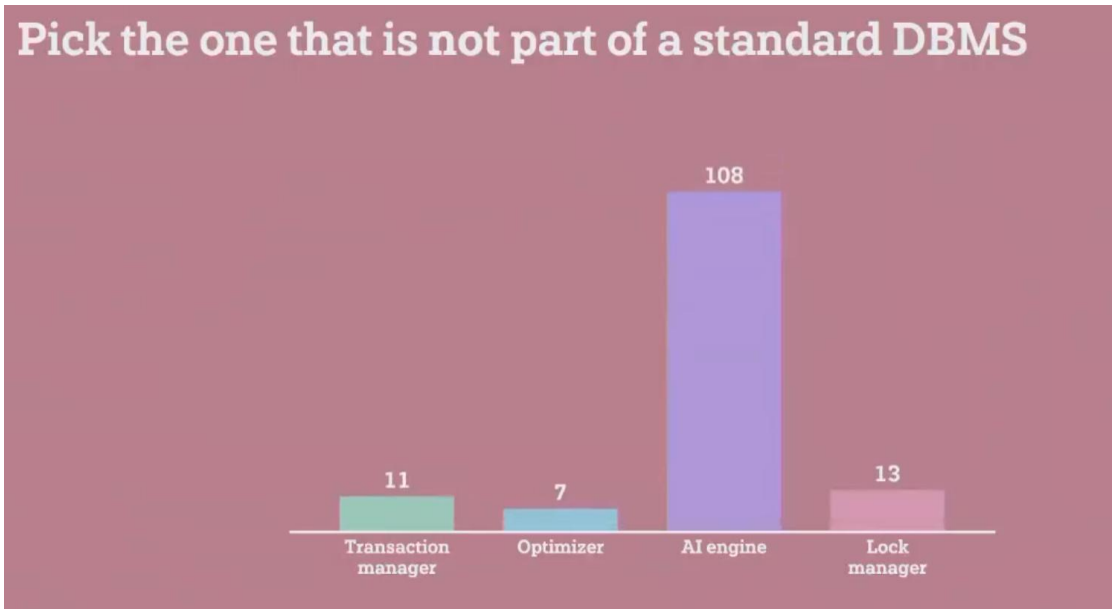
- NULLIF()

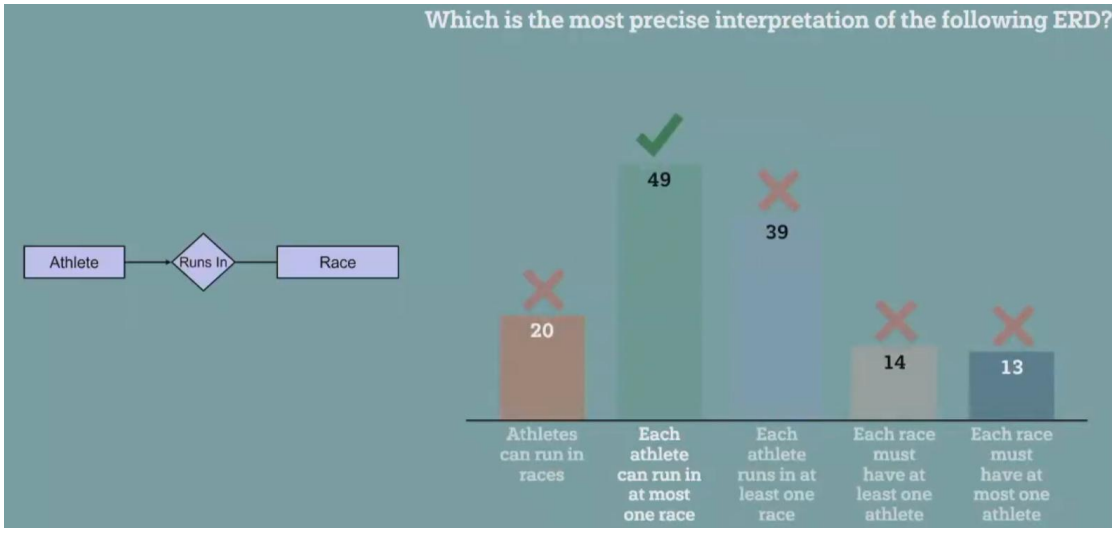
```
-- 注意是NULLIF 不是 IFNULL，并且他是把输出转换成NULL的

SELECT film_id, title, release_year, NULLIF(rating, 'TBD') rating
FROM Film
ORDER BY release_year DESC, title ASC;
```

- STRING_AGG(expression, separator ORDER BY attributes) 他就是一个聚合函数，和 MAX()之类的并无差别

Menti

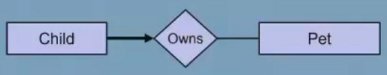
Week1


Pick the one that is not part of a standard DBMS
11
7
108
13
Transaction manager
Optimizer
AI engine
Lock manager

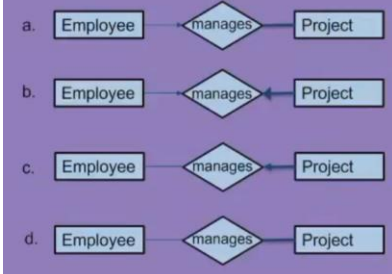

Which is the most precise interpretation of the following ERD?
Athlete
Runs In
Race
20
49
39
14
13
Athletes can run in races
Each athlete can run in at most one race
Each athlete runs in at least one race
Each race must have at least one athlete
Each race must have at most one athlete


Which is the most precise interpretation of the following ERD?
Person
Likes
Colour
21
53
9
47
Each person must like exactly one colour
Each colour must be liked by at least one person
Each person must like a different colour
Each person must like at least one colour

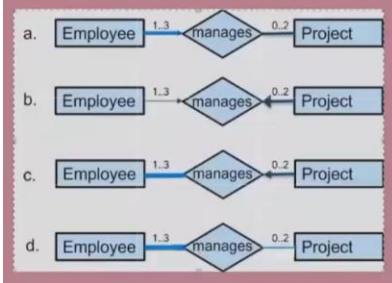
Which is the most precise interpretation of the following ERD?

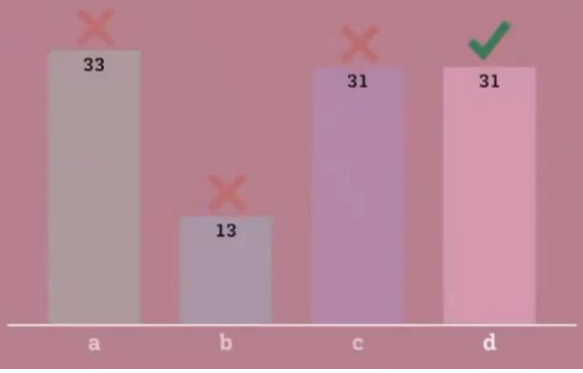


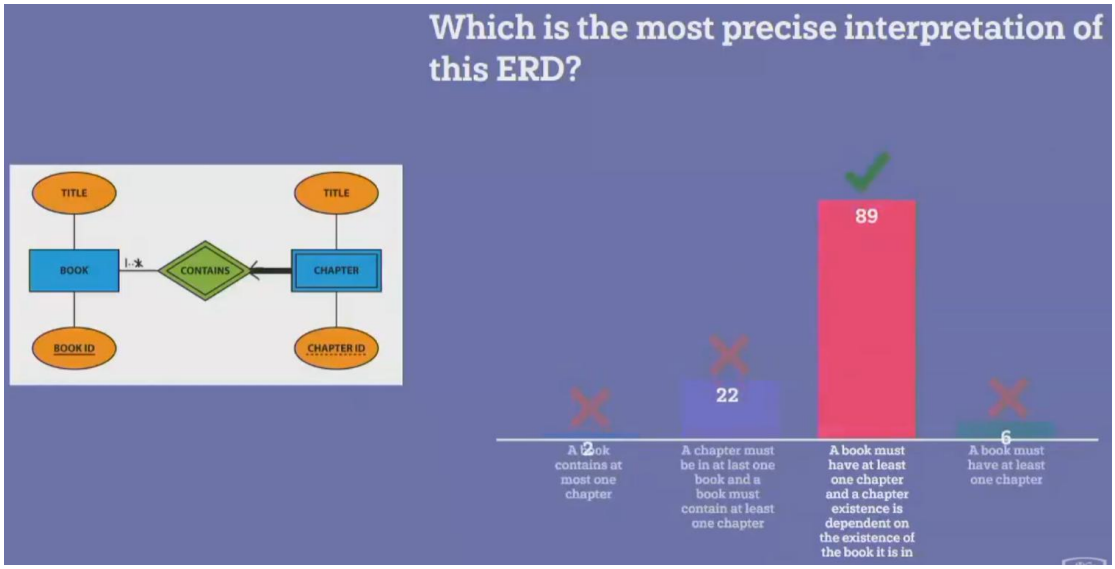
Which is the correct model for "One employee manages at most one project. Every project must be managed by at least one employee"?



Which is the correct model for "Each employee manages between 1 and 3 projects. Each project has either 0, 1, or 2 managers "?

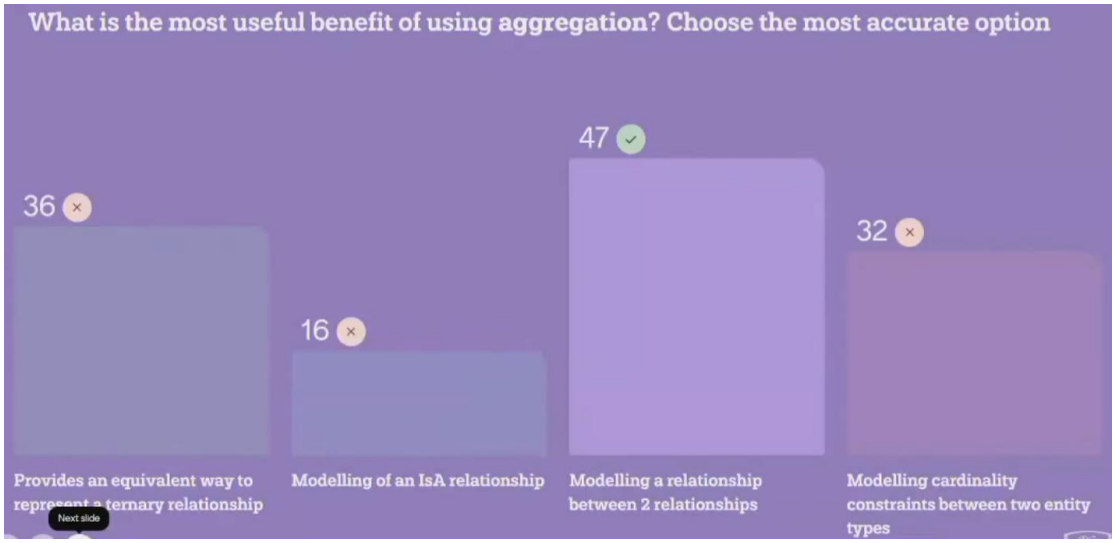


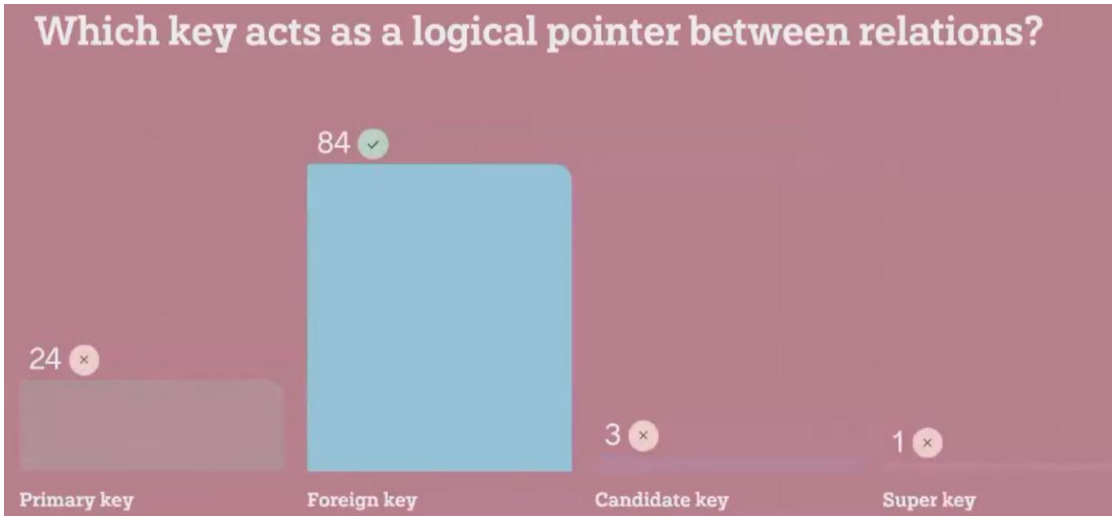


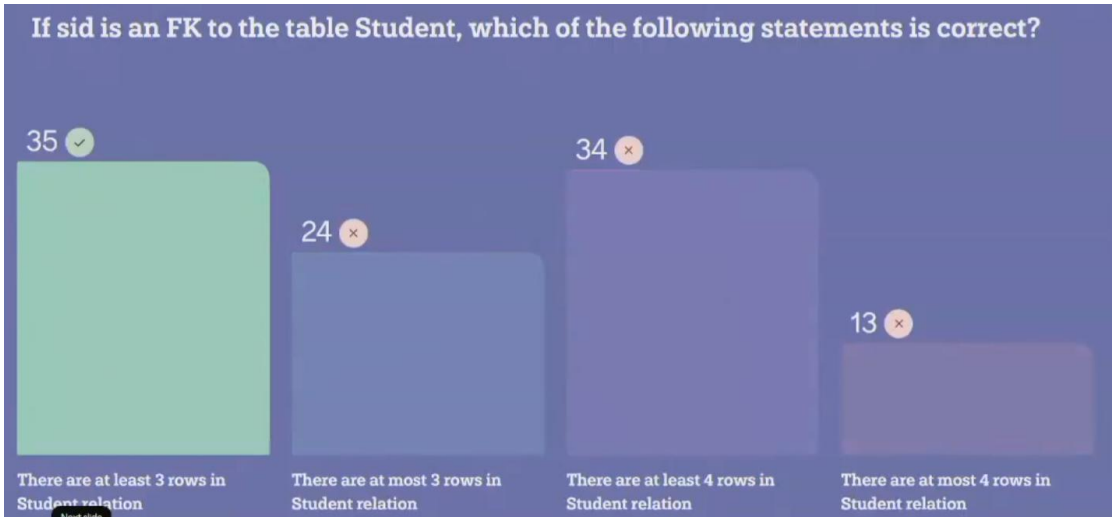

Which is the most precise interpretation of this ERD?
TITLE
BOOK
CONTAINS
CHAPTER
TITLE
BOOK ID
CHAPTER ID
2
22
89
6
A book contains at most one chapter
A chapter must be in at last one book and a book must contain at least one chapter
A book must have at least one chapter and a chapter existence is dependent on the existence of the book it is in
A book must have at least one chapter

Week 2

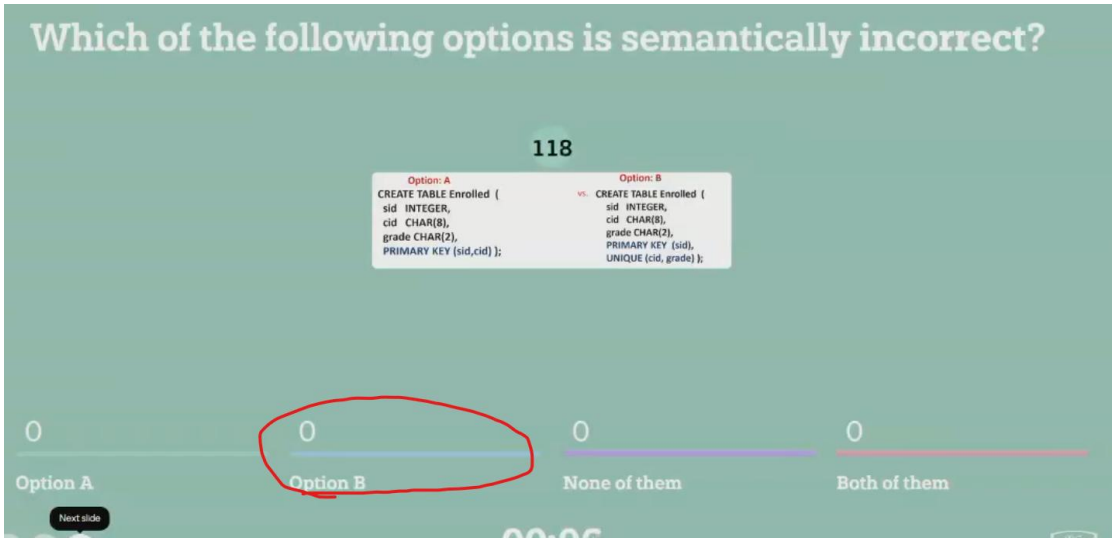

Choose the most accurate definition of a total disjoint Is-A relationship
42
33
36
35
An entity of the superclass can be in one or more subclasses
An entity of a subclass may or may not be in the superclass
Each entity of the superclass must be in one or more subclasses
Each entity of the superclass should be in exactly one subclass


What is the most useful benefit of using aggregation? Choose the most accurate option
36
16
47
32
Provides an equivalent way to represent a ternary relationship
Modelling of an IsA relationship
Modelling a relationship between 2 relationships
Modelling cardinality constraints between two entity types
Next slide

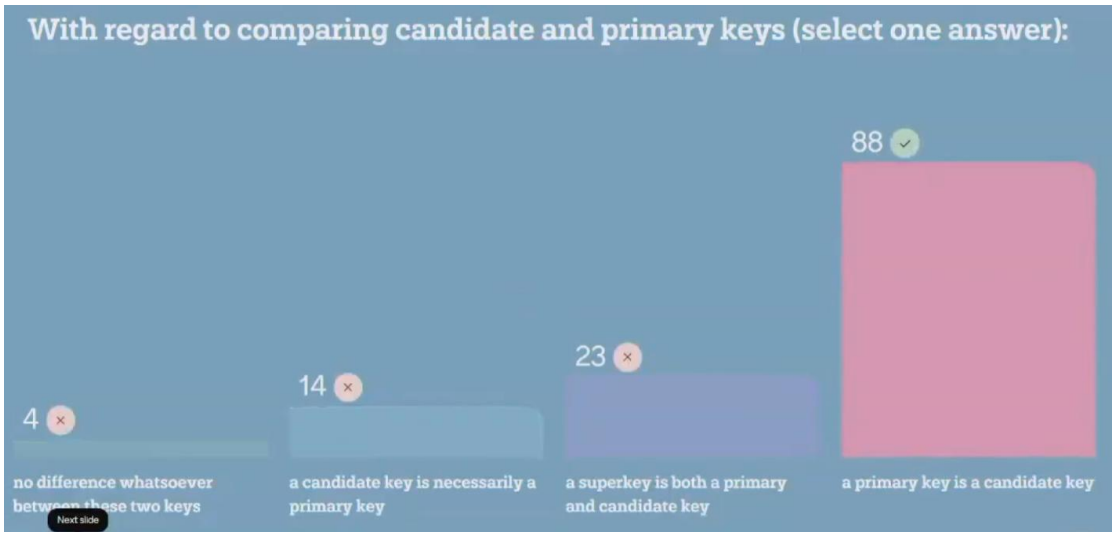




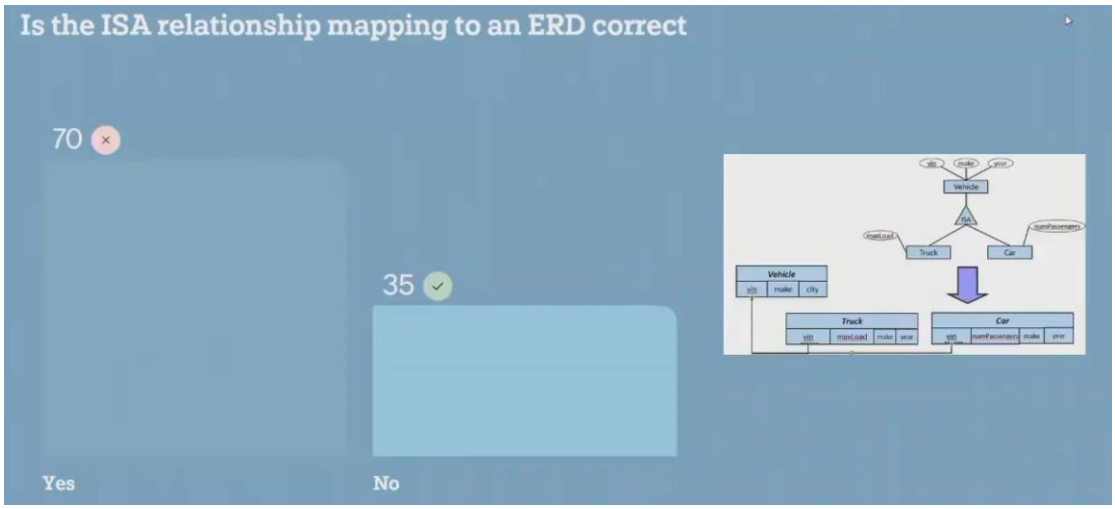
下面这个比较重要



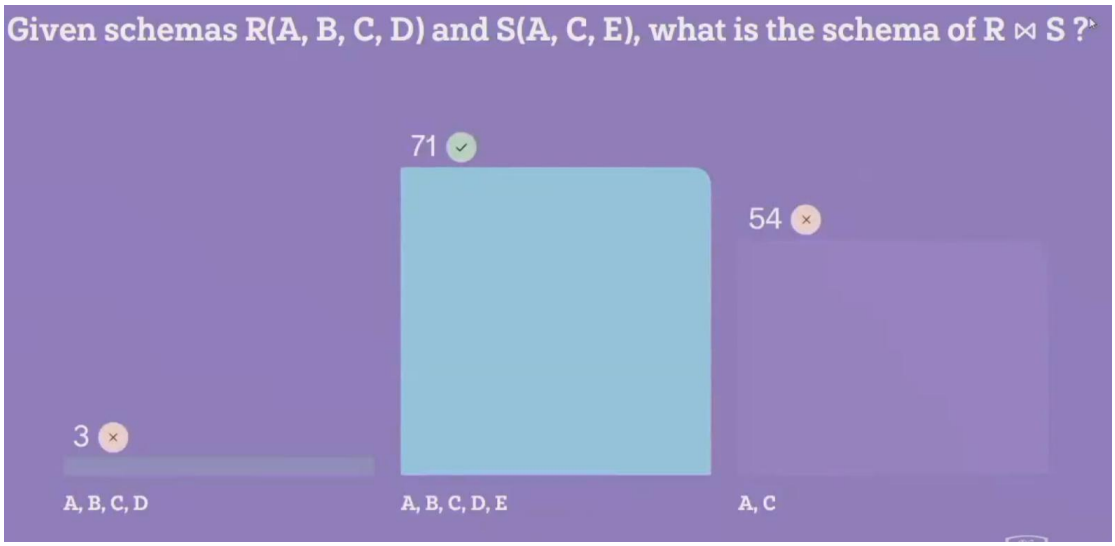
不对是因为 enroll 表必须用 studentid 和 courseid 作为主键才符合 semantical def

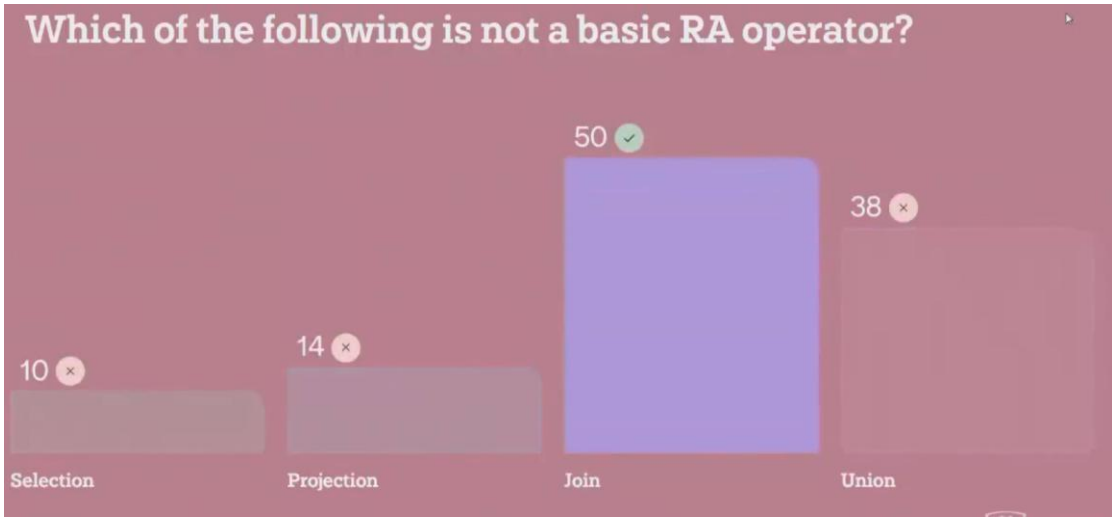


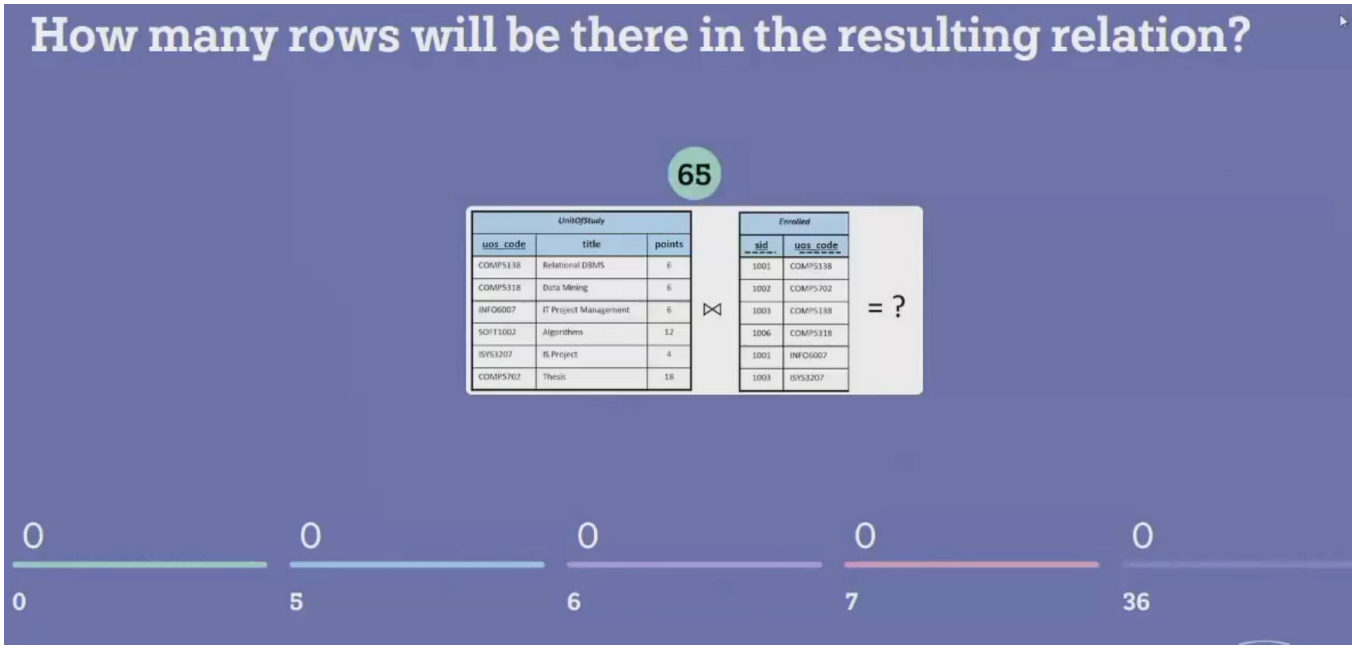
Week3



这里因为 child 包含了 parent 的 attributes 所以不对







这里我们用 uos_code 作为 natural join。Natural join 返回的是个 set，不是 table，因此不需要有 PK

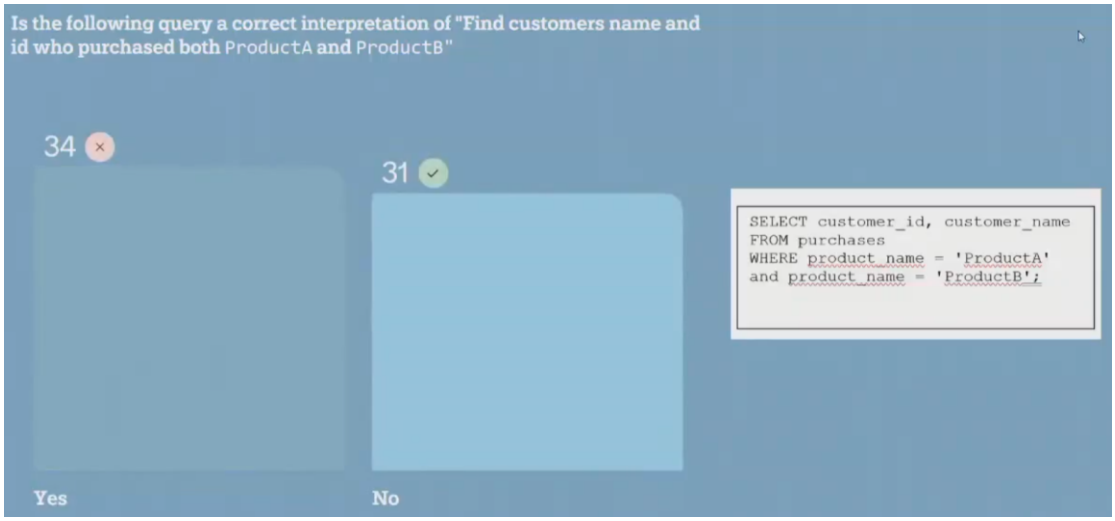
左边的 soft1002 没有对应的所以不显示

Comp5138 对应的有 2 个，所以显示两个

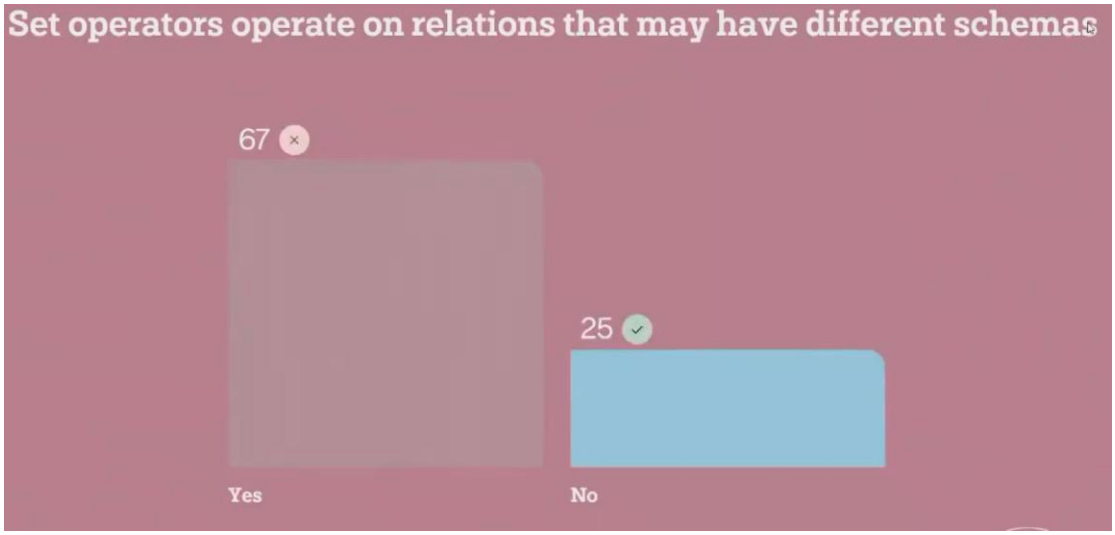
别的都是一一对应

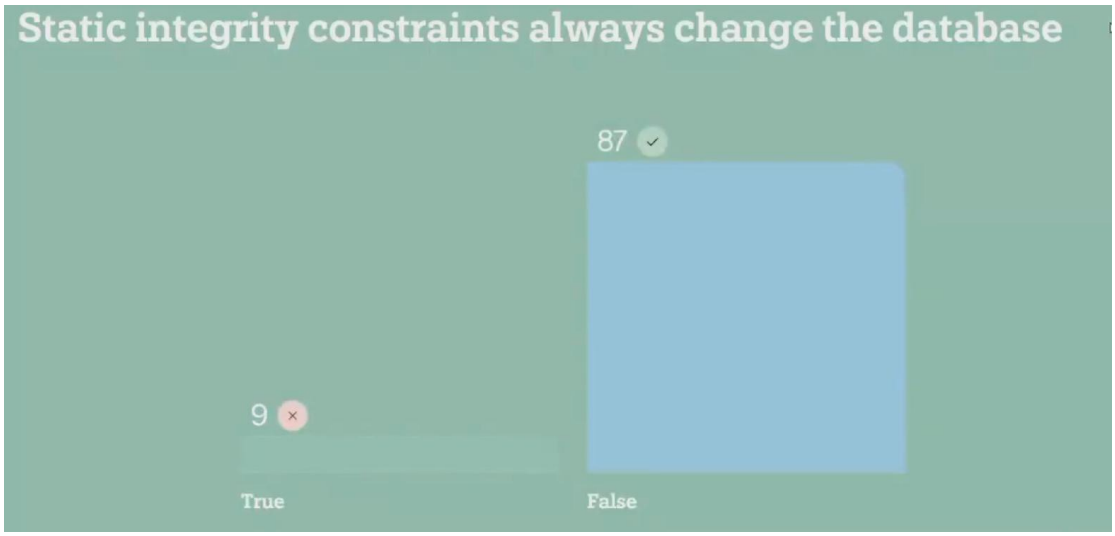
总共应该显示 6 行

Week5

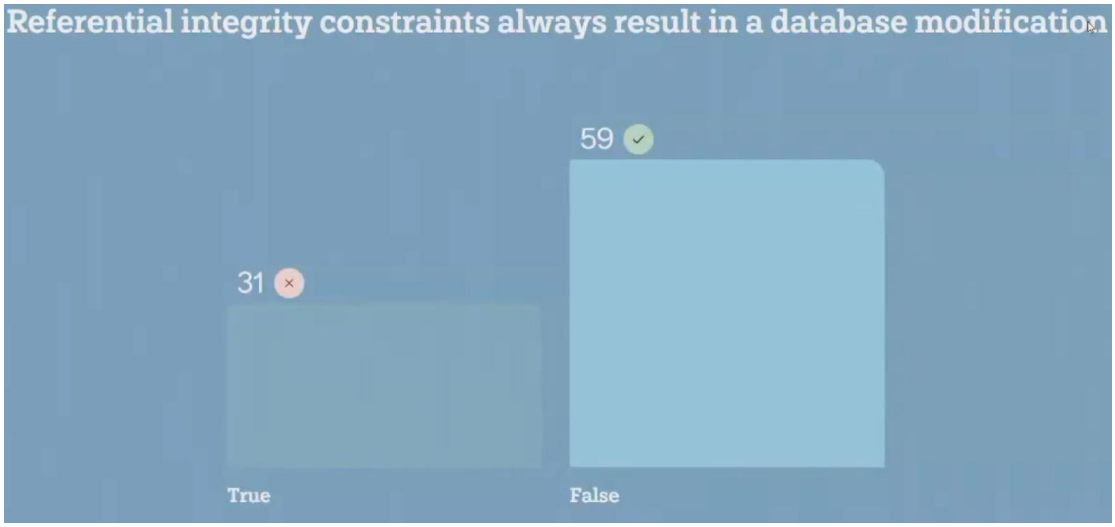


上面的思路是用 intersect 或者 subquery，参考 TUT6



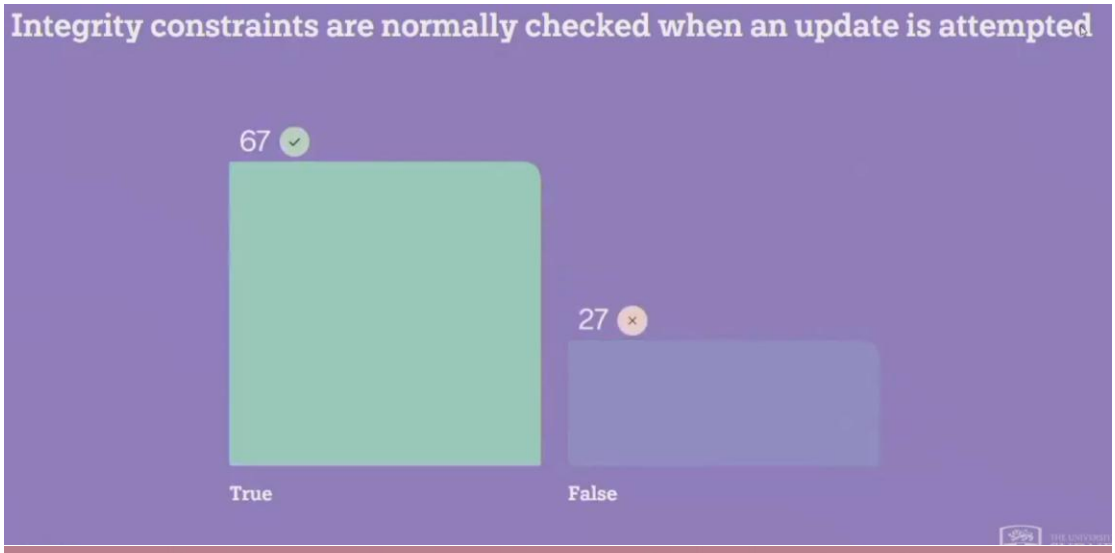


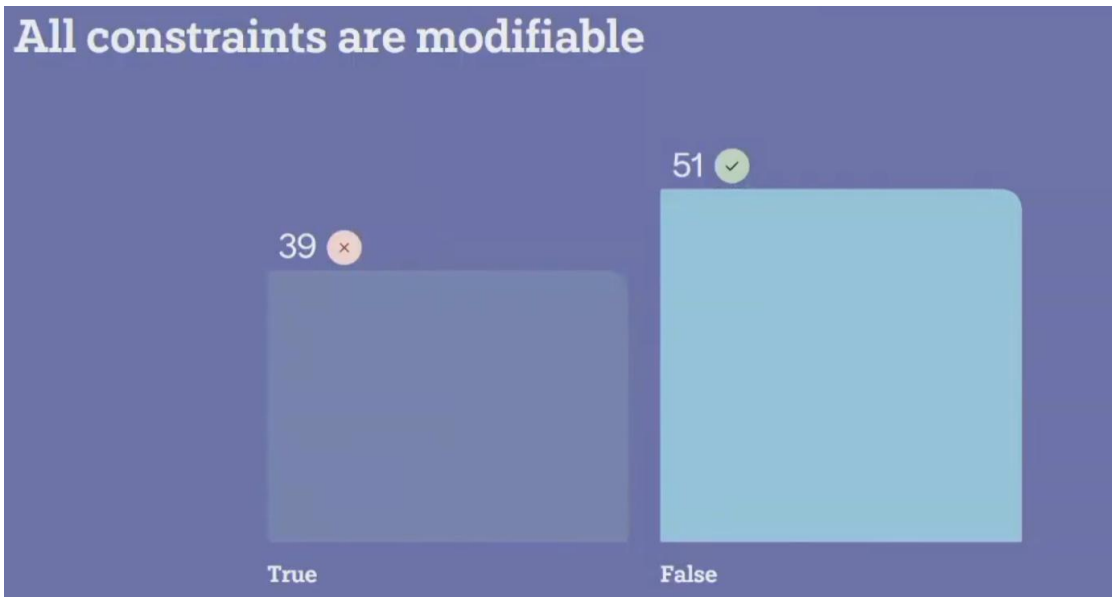
即使是 dynamic 也不应该 always，而只有在 trigger 中包含 update 的才会改变 DB



Referential Integrity：

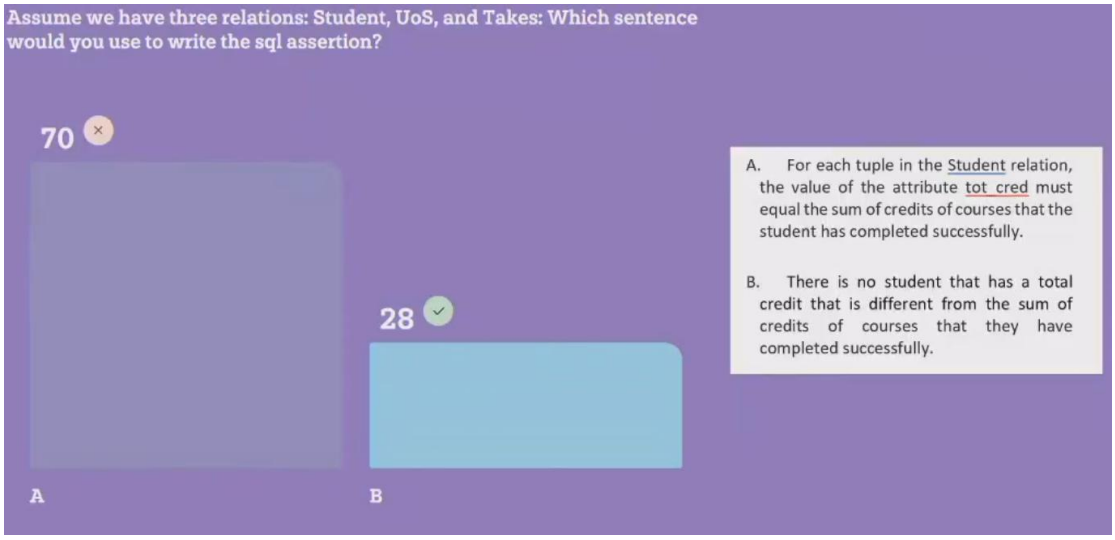
只是表达 references，对于 CASCADE, SET NULL, SET DEFAULT 是额外选项，因此不是 always



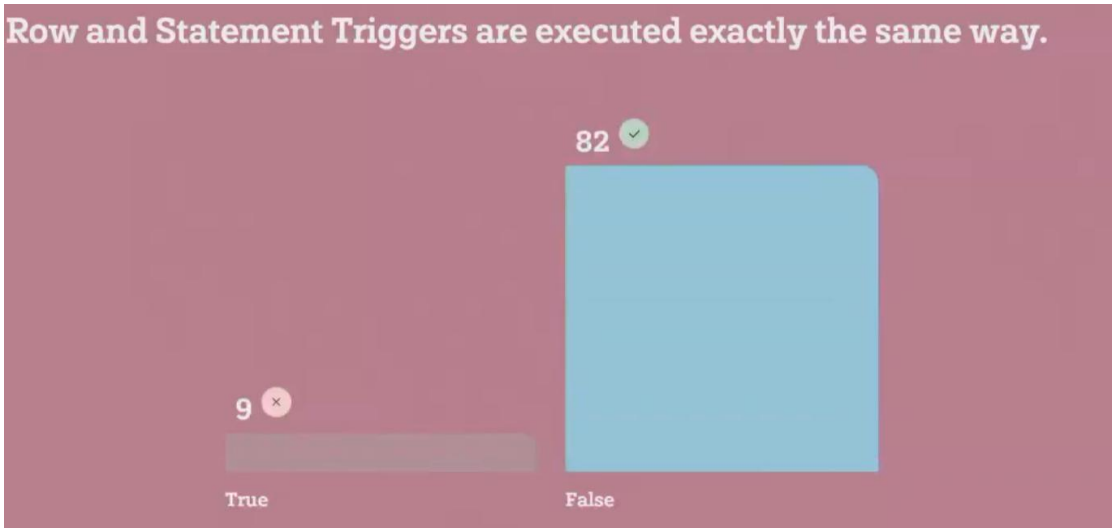


Only Domain constraints are modifiable

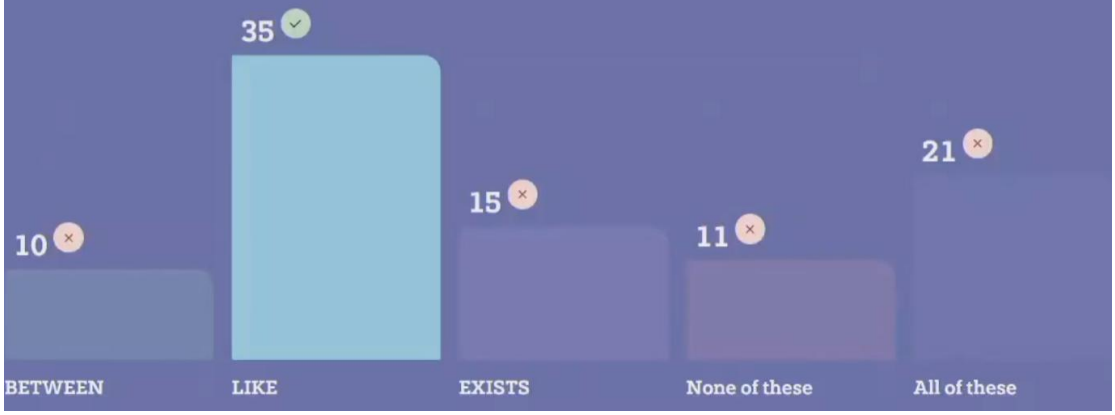
Week6



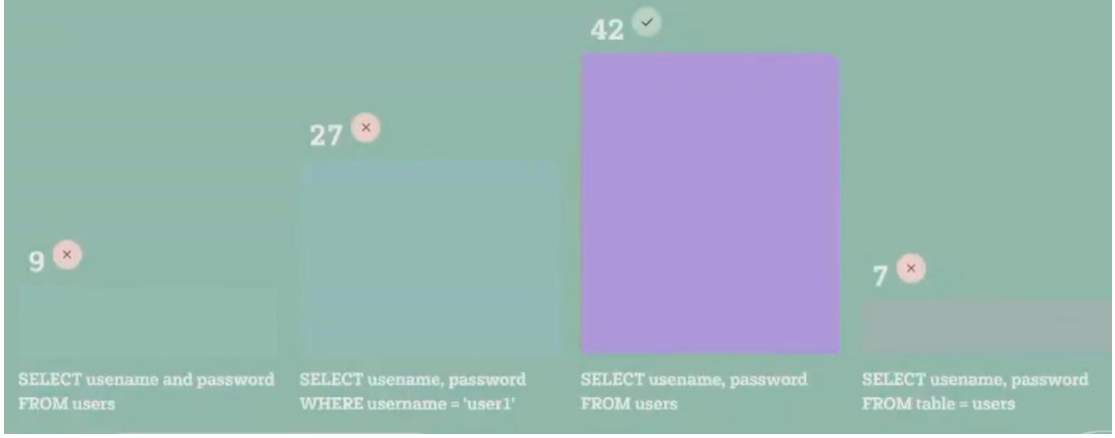
这里是因为 exists 只能判断有一个 row 满足，而不能判断所有 row 都满足



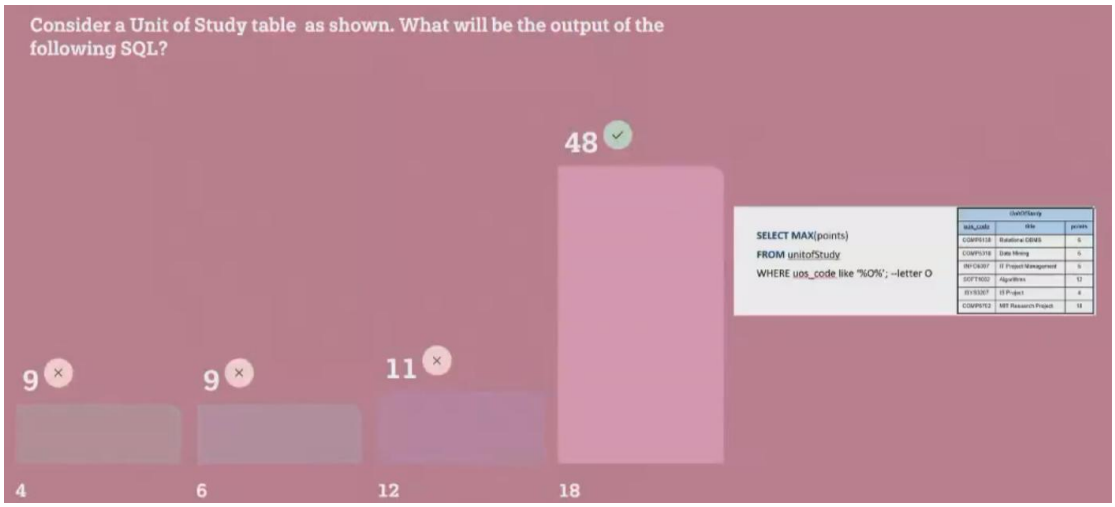
Which operator performs pattern matching?



Given a USERS table, which of the following SQL syntax is correct?

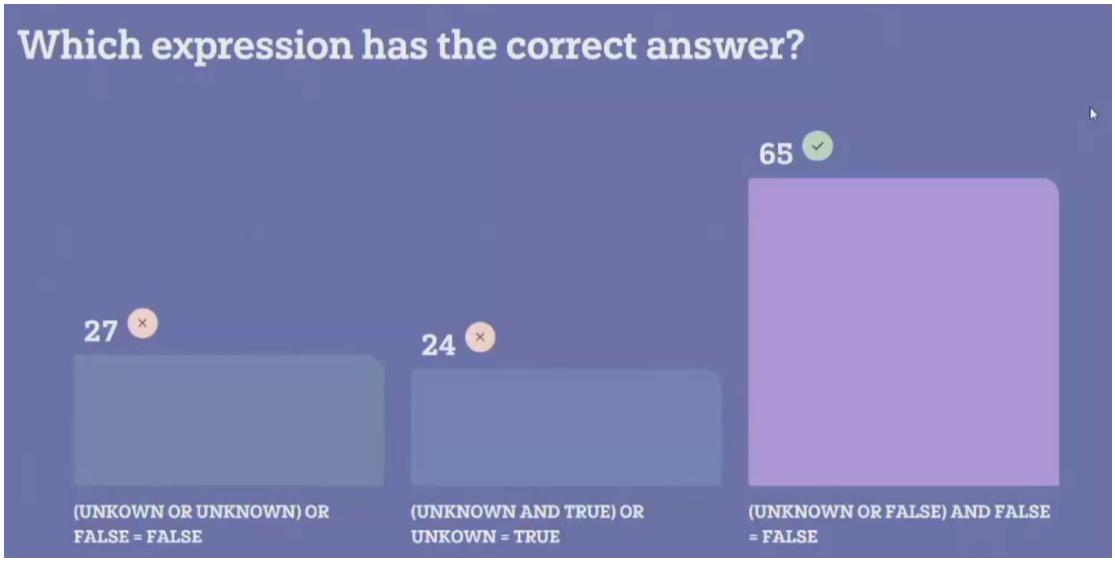


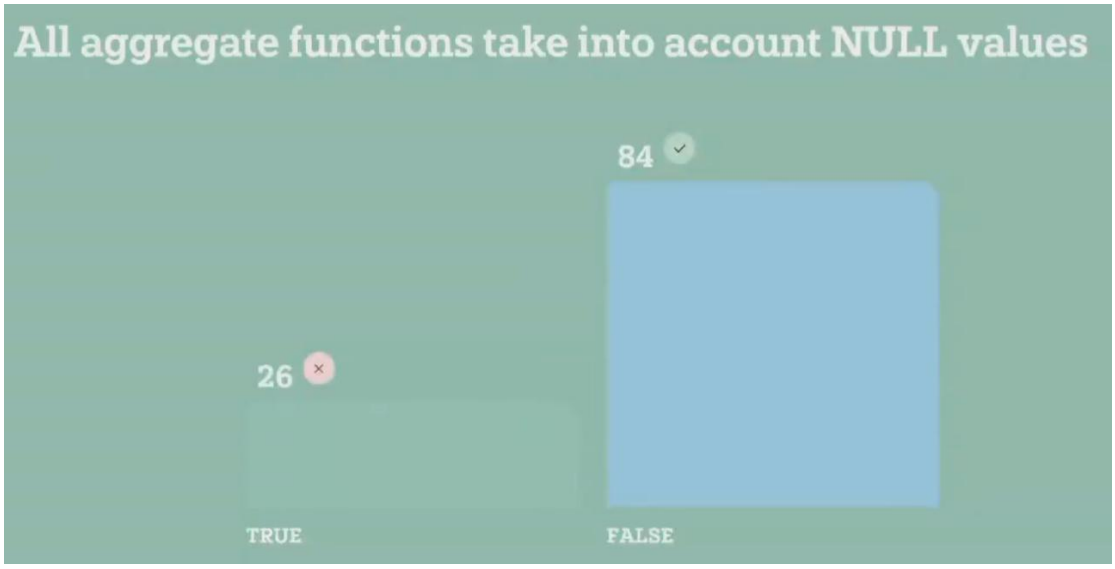
Consider an Employee table that has 10 records. It has a non-null SALARY column which is UNIQUE. What will be the output of the following SQL?


Consider a Unit of Study table as shown. What will be the output of the following SQL?
48
9
9
11
SELECT MAX(points)
FROM unitofStudy
WHERE uos_code like '%O%'; --letter O
4
6
12
18

Week7

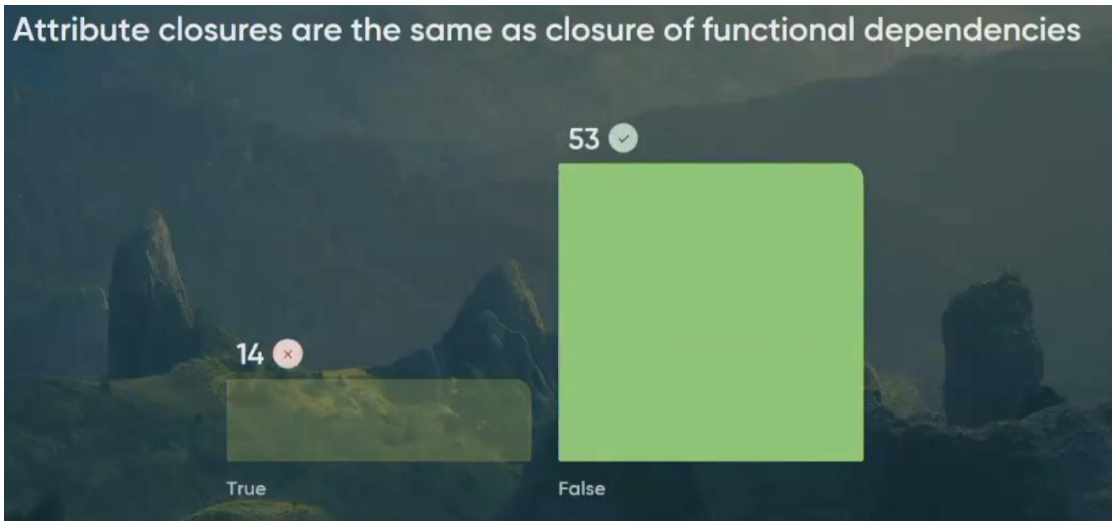

Which comparison results are correct?
47
38
31
(0 <= NULL) returns Unknown
(NULL = NULL) returns True
(NULL <= NULL) returns Unknown

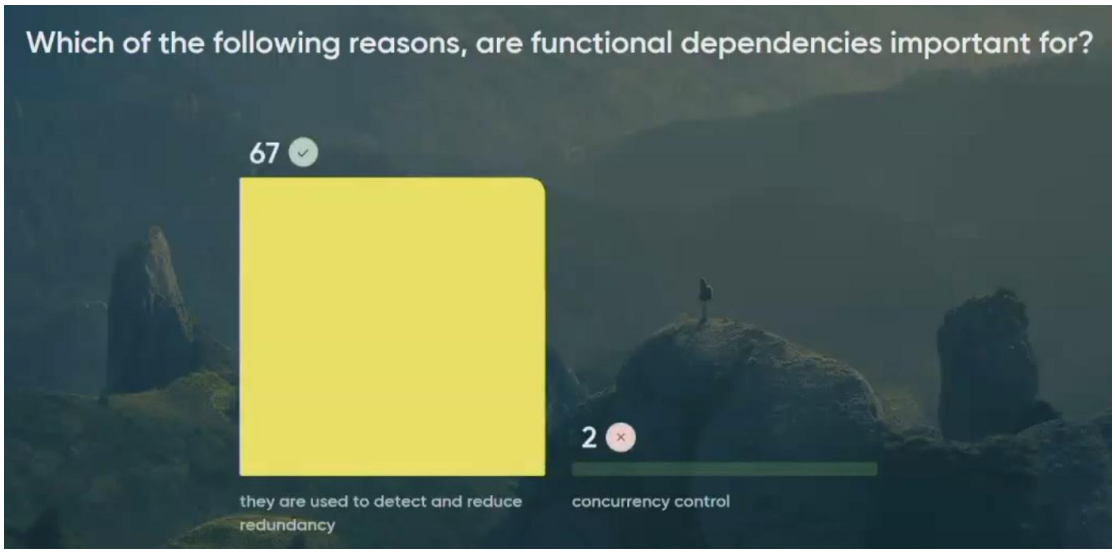

Which expression has the correct answer?
65
27
24
(UNKOWN OR UNKNOWN) OR FALSE = FALSE
(UNKNOWN AND TRUE) OR UNKOWN = TRUE
(UNKNOWN OR FALSE) AND FALSE = FALSE


All aggregate functions take into account NULL values
84
26
TRUE
FALSE

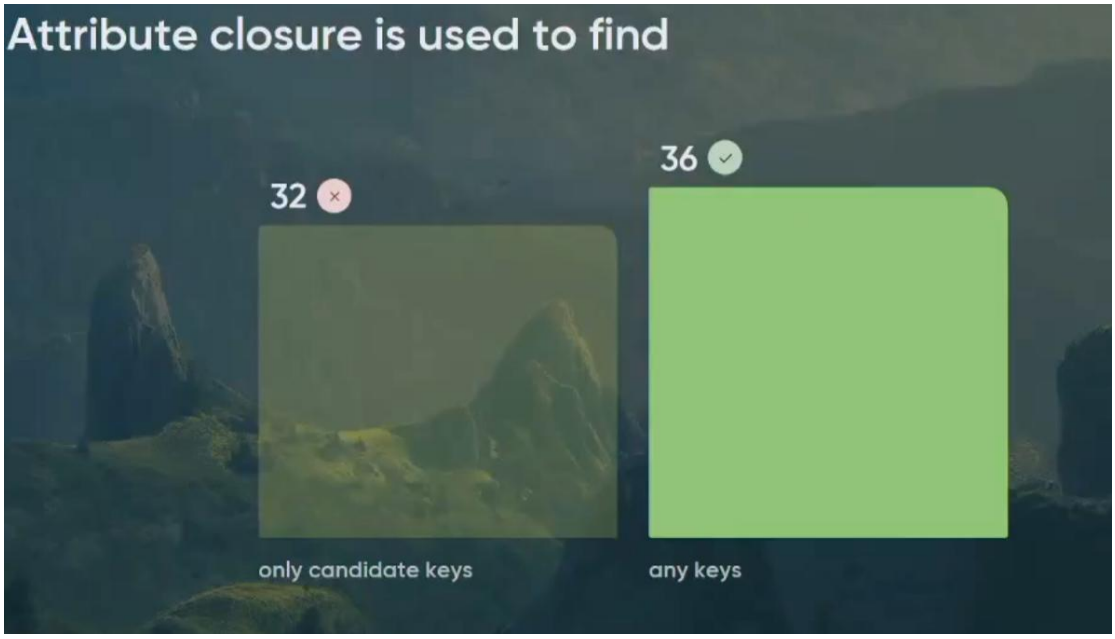
Week7


Functional dependencies
59
8
define an N-M relationship between two sets of attributes
define an N-1 relationship between two sets of attributes

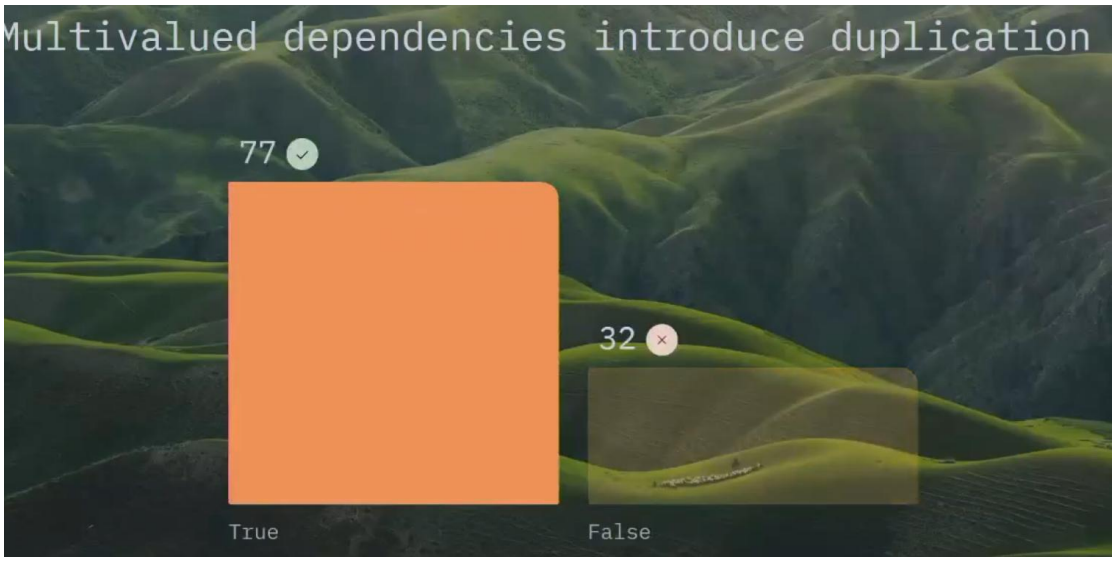

Attribute closures are the same as closure of functional dependencies
53
14
True
False



Isolation level 正是数据库中用于确保 **并发控制（Concurrency Control）** 的关键机制。



Week9


Multivalued dependencies introduce duplication
77
32
True
False

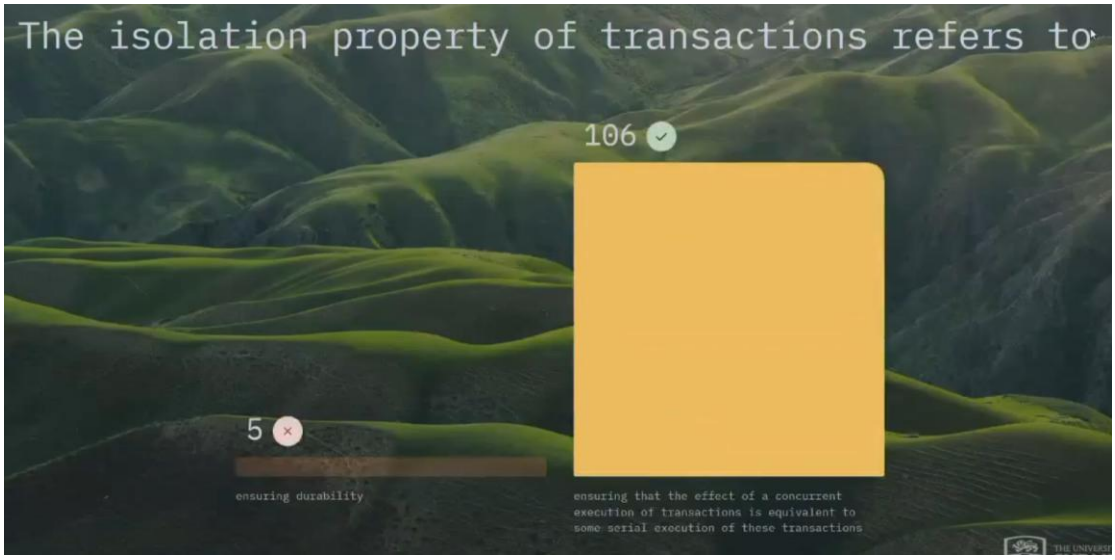

For a decomposition to be lossless-join
78
31
it needs to preserve the functional dependencies
joining the resulting decomposed relations would yield the original relation


Multivalued dependencies are needed to
67
41
to represent a relationship between any two set-valued attributes
establish independence between two relationships


Transaction consistency is important because
108
3
the database design depends on it
it defines the rules, including the integrity constraints, that need to be adhered to for the database to be in a correct state


Transaction logs are used for
94
14
auditing purposes
ensuring durability


Lost/temporary/incorrect summary problems occur in
87
24
serial executions of transactions
concurrent executions of transactions


The isolation property of transactions refers to
106
5
ensuring durability
ensuring that the effect of a concurrent execution of transactions is equivalent to some serial execution of these transactions

Example

```sql
DELETE FROM Language WHERE name = 'German';
```

```sql
INSERT INTO Actor VALUES (4711, 'Arnold', 'Schwarzenegger', 'AT');
```

```sql
UPDATE Film
SET rental_rate = rental_rate *1.2
WHERE title = 'ANGELS LIFE';
```

```sql
SELECT COUNT(*)
FROM Film
WHERE length > 180;
-- 但是WHERE length / 3.0 > 60; 就可以，因为这样的除法是小数，不会进行下面的整除

-- length / 3 用整数除法，直接把结果向下取整！181 / 3 = 60.333...

-- 整数除法结果是 60 （小数部分被直接截断）

-- 60 > 60 是 ❌不成立！
```

也就是说在这里，如果用 length/3，那么当 length = 181 的时候就不会算入，因为 181/3=60

```sql
-- 记得在select的时候表明是从哪个表里获得的字段
-- FROM 表在outer join的时候需要注意顺序

SELECT f.title, COALESCE(a.first_name, 'N/A') actor_firstname, COALESCE(a.last_name, 'N/A') actor_lastname
FROM Film f
LEFT JOIN Film_Actor fa
ON f.film_id = fa.film_id
LEFT JOIN Actor a
ON fa.actor_id = a.actor_id
WHERE f.release_year = 2004;
```

```sql
-- 注意union的也可以被ORDER BY
-- 只用在末尾添加;
SELECT name
FROM Country
UNION
SELECT name
FROM Language

ORDER BY name;
```

Write an SQL query that finds all films (by film_id and title) categorised as both “Drama” and “Family” film.

```
-- 这个很重要
SELECT f.film_id, f.title
FROM Film f
JOIN Film_Category fc ON f.film_id = fc.film_id
JOIN Category c ON fc.category_id = c.category_id
WHERE c.name = 'Drama'

INTERSECT

SELECT f.film_id, f.title
FROM Film f
JOIN Film_Category fc ON f.film_id = fc.film_id
JOIN Category c ON fc.category_id = c.category_id
WHERE c.name = 'Family';
```

在 Cartesian 里面，自己是会和自己生成 row 的

```
-- 需要考虑到自己和自己是会生成一个row的，因此要排除掉
-- 比如下面就是通过 a1.actor_id <> a2.actor_id
SELECT a1.actor_id, a1.first_name, a1.last_name
FROM Actor a1, Actor a2
WHERE a1.first_name = a2.first_name
AND a1.last_name = a2.last_name
AND a1.actor_id <> a2.actor_id
ORDER BY first_name, last_name, actor_id;
```

```
-- UTC是基准，比如中国的UTC + 8
-- SELECT EXTRACT(TIMEZONE FROM TIMESTAMP '2025-05-04 12:00:00+08:00'); 像这里是28800 秒 对应的是 8 小时
-- SELECT EXTRACT(TIMEZONE_HOUR FROM TIMESTAMP '2025-05-04 12:00:00-08:00'); -8 因为小时部分为-8
-- SELECT EXTRACT(TIMEZONE_MINUTE FROM TIMESTAMP '2025-05-04 12:00:00+08:00'); 0 因为分钟部分为0

-- 除了上面的语法外，还需要记住如何用format string来表示它们
/*
'YYYY': 四位数年份（如: 2025）

'MM': 两位数月份（如: 05）

'DD': 两位数日期（如: 04）

0 在格式化字符串中表示填充字符，用于确保数字的位数达到指定长度。
s 表示数字的正负符号
FM: 去除填充字符。确保在格式化过程中不会添加前导零或空格。


TO_CHAR(5, 'FM00') 会返回 05，即使数字是单一数字，FM 也会强制将它变成两位数字。
TO_CHAR(5, 'FM00x') 会返回 05x，因为x并不会被识别成填充符号。
TO_CHAR(-5, 'S00') 会返回 -05

和select用
SELECT TO_CHAR(CURRENT_DATE, 'YYYY');
*/

SELECT ('UTC' ||
        TO_CHAR(EXTRACT(TIMEZONE_HOUR FROM CURRENT_TIMESTAMP), 'S00') ||
        ':' ||
        TO_CHAR(EXTRACT(TIMEZONE_MINUTE FROM CURRENT_TIMESTAMP),'FM00')
) tz_offset;
```

```
SELECT film_id, title
FROM Film
-- 聚合函数不能直接用于 WHERE 子句中，需要用subquery替代
-- 聚合函数可以用于HAVING
WHERE rental_rate = (SELECT MIN(rental_rate) FROM Film) -- subquery不用;
AND release_year = 2006
ORDER BY title;
```

```
-- USING 自动合并USING()里的列名，也就是说不会产生歧义
SELECT film_id, title, actor_id, first_name, last_name
  FROM Film JOIN Film_Actor USING (film_id)
            JOIN Actor      USING (actor_id)
 WHERE film_id IN (
   SELECT film_id
     FROM Film_actor JOIN Actor USING (actor_id)
    WHERE first_name = 'PENELOPE' AND
          last_name  = 'GUINESS'
 );
```

`mod(a, b)`

Computes the remainder of `a / b`.

`round(n, d)`

Rounds `n` to `d` decimal places.

`trunc(n, d)`

Truncates `n` to `d` decimal places.

`ceil(n)`

Computes the smallest integer value not less than `n`.

`floor(n)`

Computes the largest integer value not greater than `n`.

`abs(n)`

Computes the absolute value of `n`.

Here's an example of the above functions:

▶ Run

```
1 SELECT mod(15, 4), round(15.67, 1),
2        trunc(15.67, 1), ceil(15.67),
3        floor(15.67), abs(-121.4)
```

可以在 select 和 where 里面使用

ROUND()是四舍五入，对于 150/60 来说，结果为 2；但是对于 ROUND(150/60,0)来说就是 3，对于 ROUND(150/60)来说就是 2

```
select 150/2, ROUND(150/60.0), ROUND(150/60)
```

?column?	round	round
75	3	2

如果不想四舍五入就用 TRUNC()

```sql
-- create table的;在括号外面
CREATE TABLE Film_Screening(
    screening_id INT PRIMARY KEY,
    film_id INT NOT NULL,
    theatre VARCHAR(50),
    start_time TIMESTAMP NOT NULL,
    film_start TIME,
    last_tickets DATE,

    -- 最后指定的column需要带上括号
    -- 主表和附表都要指定column
    -- 当前表不用表明                    这里要带
    FOREIGN KEY (film_id) REFERENCES Film (film_id)

);
```

```sql
SELECT COUNT(*) as count
FROM Film
-- EXTRACT里面是from不是逗号
-- 年份不确定的话，最后一个个数，25 24 23 22 21 20 19 18 17 16 而25-10 = 15
-- 还要保证年份不能超过当前
WHERE release_year >= EXTRACT(YEAR FROM CURRENT_DATE) - 9 AND release_year <= EXTRACT(YEAR FROM CURRENT_DATE);
```

```sql
SELECT COUNT(*)
FROM Film
-- 用DATE '' 可以强转数据类型为日期类
-- 需要用()来隔开string concatenate
WHERE EXTRACT(DOY FROM DATE (release_year || '-12-31')) = 366;
```

```sql
/*
    外层的 WHERE release_year = 2000 是第一道筛选;

    然后 EXISTS(...) 是逐行对外层每条记录进行条件验证;
*/
SELECT film_id, title
FROM Film f1
WHERE release_year = 2000
AND EXISTS(
    SELECT *
    FROM Film f2
    JOIN Film_Actor fa USING (film_id)
    WHERE f1.film_id = f2.film_id
)
ORDER BY title;
```

```sql
SELECT release_year, COUNT(*) number_of_films
FROM Film
GROUP BY release_year
-- chronologically 按时间顺序
-- DESC要放在后面
ORDER BY number_of_films DESC, release_year;
```

```sql
-- COUNT(*) 会计算NULL，因此这里要用COUNT(film_id)，否则那些没有对应电影的category也会有count为1

SELECT category_id, name, COUNT(film_id) count
FROM Category
LEFT JOIN Film_Category USING (category_id)
GROUP BY name, category_id
ORDER BY count DESC, name ASC;
```

```sql
SELECT film_id, title, COUNT(actor_id) count
FROM Film
JOIN Film_Actor USING (film_id)-- USING 要带上括号
GROUP BY film_id, title
-- HAVING 子句必须使用完整的聚合表达式，不能直接引用 SELECT中的alias;
-- HAVING可以使用非聚合函数，比如actor_id > 5
HAVING COUNT(actor_id) >= 5
ORDER BY title;
```

```sql
SELECT c.name country, COUNT(actor_id) num_actors
FROM Actor a
JOIN Country c
ON a.nationality = c.short_code
GROUP BY country -- GROUP BY能用SELECT中的alias
ORDER BY num_actors DESC, country
LIMIT 5;

/*
GROUP BY能用SELECT中的alias, 下面这种cancate的也可以

SELECT (c.name || c.name) country, COUNT(actor_id) num_actors
FROM Actor a
JOIN Country c
ON a.nationality = c.short_code
GROUP BY country -- GROUP BY能用SELECT中的alias
ORDER BY num_actors DESC, country
LIMIT 5;
*/

-- ordinal ranking: 每个条目都有独立编号，即使有相同值，也不会并列; ROW_NUMBER()
-- dense ranking: 如果有相同值，会并列排名，下一个排名不会跳过; DENSE_RANK()
-- Standard ranking (或叫 sparse ranking): 有相同值时会并列，下一个排名会跳过; RANK()

SELECT first_name, last_name, COUNT(film_id) films
FROM Actor
JOIN Film_Actor USING (actor_id)
WHERE nationality = 'US'
GROUP BY actor_id, first_name, last_name -- 注意！！名字组合可能不唯一，因此要用actor_id来区分
ORDER BY films DESC, first_name, last_name
LIMIT 10;


-- rank() OVER (ORDER BY mark DESC) 是窗口函数，只是生成了一列排名，不影响结果行的顺序。
-- RANK()给的是Standard ranking
SELECT RANK() OVER (ORDER BY COUNT(film_id) DESC) rank, name category, COUNT(film_id) films
FROM Category
JOIN Film_Category USING (category_id)
GROUP BY name
ORDER BY rank, name
LIMIT 5;
```

```sql
-- 只有date可以直接运算，而interval只是一段时间

SELECT film_id, title, rental_rate, replacement_cost,
    GREATEST(rental_rate*(DATE '2016-04-12' - DATE '2016-03-24'),
    replacement_cost) late_fee
FROM Film f
JOIN Film_Actor fa USING (film_id)
JOIN Actor a USING (actor_id)
WHERE (a.first_name || ' ' || a.last_name) = 'JOE SWANK'


-- STRING_AGG(expression, separator ORDER BY attributes)
-- 他就是一个聚合函数，和MAX()之类的并无差别

SELECT film_id, title, STRING_AGG(name, ',' ORDER BY name) cat_agg
FROM Film
JOIN Film_Category USING (film_id)
JOIN Category USING (category_id)
GROUP BY film_id, title;


/*
CREATE DOMAIN GradingScheme
       CHAR(2) CHECK (value in ('FA','PA','CR','DI','HD'));
创建自定义数据类型
*/
CREATE DOMAIN Distance INTEGER CHECK (value BETWEEN 0 AND 25000);

CREATE TABLE Airplane(
       model VARCHAR(10) PRIMARY KEY,
       manufacturer VARCHAR(50) NOT NULL,
       flight_range Distance,
       cruise_speed INT,
       enginetype VARCHAR(50) DEFAULT 'turbofan',
       first_flight DATE,

       -- check condition需要括号包裹
       CHECK (enginetype in ('turbofan', 'turbojet', 'turboprop')),
       CHECK (cruise_speed > 200)
       -- BETWEEN不能写成25000 AND 0
       -- CHECK (flight_range BETWEEN 0 AND 25000)
);
```

```
/*
记住 ON DELETE/UPDATE
        CASCADE
        SET NULL
        SET DEFAULT
两个可以同时存在，比如 ON DELETE .... ON UPDATE ....
*/
CREATE TABLE Inventory(
    inventory_id INT PRIMARY KEY,
    film_id SMALLINT NOT NULL,
    store_id INT,
    quantity INT,
    last_update TIMESTAMP NOT NULL,

    --  FK前面这里也要带上括号
    FOREIGN KEY (film_id) REFERENCES film (film_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (store_id) REFERENCES Store (store_id) ON DELETE SET NULL ON UPDATE CASCADE,
    CHECK (quantity >= 0)
);
```

```
CREATE OR REPLACE FUNCTION trg_check_vehicle_classification_fn()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS (
        SELECT 1 FROM NewCar WHERE VIN = NEW.VIN
    ) AND NOT EXISTS (
        SELECT 1 FROM PreOwnedCar WHERE VIN = NEW.VIN
    ) THEN
        RAISE EXCEPTION 'Vehicle (VIN=%) must appear in either NewCar or PreOwnedCar', NEW.VIN;
    END IF;

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

▷Run | Select
CREATE CONSTRAINT TRIGGER trg_check_vehicle_classification
AFTER INSERT OR UPDATE ON Vehicle
FOR EACH ROW
EXECUTE FUNCTION trg_check_vehicle_classification_fn();
```

```
CREATE OR REPLACE FUNCTION on_sale_must_have_listing()
RETURNS TRIGGER AS $$
BEGIN
    IF(
        NEW.Status = 'for sale'
    )
    THEN
        IF NOT EXISTS(
            SELECT 1 FROM VehicleListing WHERE VIN = NEW.VIN
        )
        THEN
            RAISE EXCEPTION 'Car % does no have vehicle listing while it is curretly for sale!', NEW.VIN;
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

▷Run | Select
CREATE CONSTRAINT TRIGGER trg_on_sale_must_have_listing
AFTER INSERT ON Vehicle
FOR EACH ROW
EXECUTE FUNCTION on_sale_must_have_listing();
```

```sql
CREATE OR REPLACE FUNCTION check_car_on_sale()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM Vehicle
        WHERE VIN = NEW.VIN AND Status = 'for sale'
    ) THEN
        -- 9
        UPDATE Vehicle
        SET Status = 'has been sold'
        WHERE VIN = NEW.VIN;
        RETURN NEW;
    ELSE
        RAISE EXCEPTION 'Vehicle with VIN % is not for sale or not inside table', NEW.VIN;
    END IF;
END;

$$ LANGUAGE plpgsql;
```

Run | Select

```sql
CREATE TRIGGER trg_check_car_on_sale
BEFORE INSERT ON Sale
FOR EACH ROW
EXECUTE FUNCTION check_car_on_sale();
```