项目六 Hive 数据查询（DQL）综合实践工作手册 — 教师答案手册

> **注意**：本文件为教师参考答案，请勿分发给学生。
>
> **使用说明**：答案基于本次实践所创建的标准数据集，部分结果可能因数据量而略有差异。

第一部分：环境准备与数据加载 — 答案

1.2 创建数据库与表结构

步骤1 — 记录结果：

> 初次创建时，unit6db数据库中没有任何表（空库），SHOW TABLES 返回空结果。

步骤3 — t_usa_covid19数据加载验证：

> 共30行数据（美国5个州30个县的新冠数据）。

步骤5 — 分区数量：

> 分区数 = 1个日期（2021-01-28）× 5个州 = 5个分区。
>
> 分区列表：
>
> - count_date=2021-01-28/state=Alabama
> - count_date=2021-01-28/state=Arizona
> - count_date=2021-01-28/state=California
> - count_date=2021-01-28/state=Illinois
> - count_date=2021-01-28/state=New York
> - count_date=2021-01-28/state=Texas
> - count_date=2021-01-28/state=Washington

步骤6 — student表记录数：

> student表共有22条记录（22名学生）。

步骤7 — 思考题：

> - employee 的 id：1201, 1202, 1203, 1204, 1206
> - employee_address 的 id：1201, 1202, 1204, 1206, 1207
> - employee_connection 的 id：1201, 1203, 1204, 1205, 1206
> - 三表共同拥有的 id：1201, 1204, 1206
>
> **影响**：由于id不完全一致，不同JOIN类型会产生不同结果——INNER JOIN只返回共同存在的id，LEFT JOIN以左表为准保留所有员工信息。这模拟了真实业务场景中数据不完整的情况。

第二部分：SELECT 基础查询 — 答案

2.1 select_expr 选择表达式

练习1：

```
SELECT * FROM t_usa_covid19_p;
```

> 查询返回所有分区的全部数据，共30行。

练习2：

```
SELECT county, cases, deaths FROM t_usa_covid19_p;
```

练习3：

设置参数后，使用 `SELECT ^c.* FROM t_usa_covid19_p LIMIT 5;`
返回的字段为：**county**、**cases**（所有以字母c开头的字段）。

练习4：

```
SELECT current_database();
```

结果是：**unit6db**

练习5：

`SELECT count(county) FROM t_usa_covid19_p;` 结果为：**30**

2.2 ALL 与 DISTINCT 去重

练习6：

查询返回30行（全部数据，每行都有state字段，含重复值）。

练习7：

是的，两次查询结果行数一致（30行）。ALL是默认行为，与省略关键字效果相同。

练习8：

DISTINCT去重后共有 **7** 个不同的州（Alabama、Arizona、California、Illinois、New York、Texas、Washington）。

练习9 — 思考题：

多字段DISTINCT的去重逻辑是：将多个字段的值组合视为一个整体进行去重。只有当county

> AND state的组合完全相同时，才被认为是重复行并删除。与单字段相比，多字段组合去重的粒度更细，保留的记录通常更多。

练习10（扩展）：

```
SELECT count(DISTINCT dept) FROM student;
```

> 结果：**3**（CS、IS、MA三个系别）

或者：

```
SELECT DISTINCT dept FROM student;
```

> 返回：CS、IS、MA

第三部分：WHERE 子句与分区裁剪 — 答案

练习1：

> 返回 **0行。**因为 `1 > 2` 是恒假表达式（false），WHERE条件过滤掉了所有行，没有任何记录满足条件。

练习2：

> 返回5行（LIMIT 5限制）。`1 = 1` 是恒真表达式（true），所有行都满足条件。
>
> **说明**：WHERE子句本质是一个布尔过滤器，只有布尔值为true的行才会被保留。

练习3：

州名长度超过10个字符的州：
- "California"（10个字符，不符合 > 10）
- "Washington"（10个字符，不符合 > 10）
- "New York" 等

实际上本数据集中：Alabama(7)、California(10)、New York(8)、Illinois(8)、Arizona(7)、Texas(5)、Washington(10)
length > 10 的州：无（或Washington=10不满足>10）

参考答案：若无结果，说明当前数据集中所有州名长度 ≤ 10。可将阈值改为 `> 7` 测试。

练习4：

返回Los Angeles（California）和Cook（Illinois）的记录，共2行。

练习5 — 错误信息：

错误关键词：`SemanticException: Not yet supported place for UDAF 'count'`

正确写法：
```sql
SELECT state, count(deaths)
FROM t_usa_covid19_p
WHERE count_date = "2021-01-28"
GROUP BY state
HAVING count(deaths) > 100;
```

练习6：

WHERE deaths > 1000（无分区限制），Hive需要扫描所有分区，返回多行数据。

练习7：

> WHERE state = 'California' AND deaths > 1000，Hive只扫描state=California的分区，大幅减少I/O量。
>
> **分区裁剪优势**：避免全表扫描，只读取相关分区的数据文件，显著提升查询性能。在大数据量场景下，性能提升尤为明显。

练习8：

> 同时指定count_date和state两个分区字段，Hive精确定位到单个分区，效率最高。
>
> **分区字段**：count_date 和 state 是分区字段，普通字段是 county、fips、cases、deaths。
> 查询时应优先使用分区字段（count_date、state）作为WHERE条件，以触发分区裁剪。

练习9（扩展）：

```
SELECT * FROM student
WHERE dept = 'CS' AND sex = '男' AND age > 19;
```

> 符合条件的学生（CS系、男生、年龄>19）：孙庆（23岁）、冯伟（21岁）

第四部分：GROUP BY 与 HAVING — 答案

练习1：

州名	县数量
Alabama	5
Arizona	3

California	5
Illinois	4
New York	5
Texas	5
Washington	3

练习2 — 错误信息：

> 错误关键词：`SemanticException: Expression not in GROUP BY key 'deaths'`
>
> **规则说明**：GROUP BY的SELECT中，字段必须满足以下之一：
> 1. 是GROUP BY的分组字段（如state）
> 2. 被聚合函数处理（如sum(deaths)、count(deaths)等）
>
> deaths既不是分组字段，也没有被聚合函数处理，因此报错。

练习3：

```
SELECT state, sum(deaths) AS total_deaths
FROM t_usa_covid19_p
WHERE count_date = "2021-01-28"
GROUP BY state;
```

> 死亡人数最多的州：**New York**（total_deaths = 38336）

练习4 — round()函数：

> `round(avg(cases), 2)` 的作用是将avg计算结果保留2位小数，避免显示过长的浮点数，使结果更易读。

练习5（扩展）：

```
SELECT dept, sex, count(*) AS stu_count, round(avg(age), 2) AS avg_age
FROM student
GROUP BY dept, sex;
```

练习6 — 错误原因：

> 错误原因：WHERE子句在GROUP BY之前执行，此时分组尚未完成，sum(deaths)等聚合函数的结果还不存在，所以无法在WHERE中引用聚合函数。

练习7：

```
SELECT state, sum(deaths) AS total_deaths
FROM t_usa_covid19_p
WHERE count_date = "2021-01-28"
GROUP BY state
HAVING sum(deaths) > 10000;
```

> 符合条件（总死亡人数 > 10000）的州：
>
> - New York（~38336）
> - California（~28705）
> - Texas（~10130，视数据而定）

练习8 — 优化写法优势：

> 使用别名在HAVING中引用（`HAVING cnts > 10000`）的优势：
>
> 1. **可读性更高**：cnts语义清晰
> 2. **避免重复计算**：Hive在GROUP BY阶段已计算出聚合结果，HAVING通过别名直接引用，不需要再次计算sum(deaths)
> 3. **代码更简洁**

练习9（综合）：

```
SELECT dept, count(*) AS stu_count, round(avg(age), 2) AS avg_age
FROM student
WHERE dept IN ('CS', 'IS')
GROUP BY dept
HAVING avg(age) > 19 AND count(*) > 3;
```

第五部分：LIMIT 与 SQL 执行顺序 — 答案

练习1：

> 加州（California）共5个县，返回5行。

练习3 — LIMIT双参数：

> `LIMIT 2, 3` 的偏移量2表示从第3行开始（索引从0开始，0=第1行，1=第2行，2=第3行），返回3条记录。

练习4（扩展）：

```
SELECT * FROM student
ORDER BY num
LIMIT 2, 3;
```

> 返回第3、4、5条学生记录（num：95003、95004、95005）。

练习5 — 执行顺序分析：

1. **FROM**：从 t_usa_covid19_p 表读取数据
2. **WHERE**：过滤出 count_date = "2021-01-28" 的行，触发分区裁剪
3. **GROUP BY + sum()**：按state分组，计算每组的sum(deaths)
4. **HAVING**：过滤保留cnts > 10000的组
5. **SELECT**：从符合条件的组中提取state和cnts列，输出最终结果

练习6 — 判断正误：

- A：❌ 错误——WHERE子句中不能使用聚合函数sum()
- B：✅ 正确——HAVING中可以使用聚合函数
- C：❌ 错误——WHERE应放在GROUP BY之前，语法顺序错误
- D：✅ 正确——使用别名cnt在HAVING中引用是合法写法

> **正确选项：B、D**

第六部分：高阶排序语法 — 答案

练习2 — 死亡人数最多的3个加州县：

> 1. Los Angeles（deaths: 17096）
> 2. San Bernardino（deaths: 2997）
> 3. Riverside（deaths: 2997）或 Orange（deaths: 2538）

练习4 — NULLS LAST含义：

> ASC升序排列时，默认NULL值排在最前面（NULLS FIRST）。指定 `NULLS LAST` 后，NULL值排在升序结果的最后面，非NULL值按升序正常排列。

练习5 — CLUSTER BY日志信息：

> 日志提示：`Number of reduce tasks not specified. Estimated from input data size: 1`
> 即Hive自动估算需要1个reducetask，数据量小时通常为1。

练习6 — 设置2个reducetask后：

> 结果分成2组（2个bucket），每组内的num字段升序排列。哈希规则：num % 2 = 0 的放入一组，num % 2 = 1 的放入另一组。

练习7 — 错误信息与原因：

错误信息：`SemanticException: Cannot have both CLUSTER BY and SORT BY clauses`

原因：CLUSTER BY 内置了分组（DISTRIBUTE BY）和排序（SORT BY）功能，这两个子句不能与CLUSTER BY同时使用。此外，CLUSTER BY要求分组字段和排序字段必须是同一个字段，且只支持升序，无法实现"按sex分组，按age降序"这种需求。

练习8 — 观察结果：

男生（sex='男'）和女生（sex='女'）分为两组，每组内按age从大到小降序排列。

DISTRIBUTE BY sex 确保相同性别的学生分配到同一个reducer，SORT BY age DESC 在每组内进行降序排序。

练习9 — 等价关系验证：

两种方式结果相同。当DISTRIBUTE BY和SORT BY字段相同时，等价于CLUSTER BY。

即：`CLUSTER BY col` = `DISTRIBUTE BY col SORT BY col`（升序）

CLUSTER BY是DISTRIBUTE BY + SORT BY（同字段升序）的简写形式。

第七部分：UNION 联合查询与子查询 — 答案

练习2 — UNION ALL结果：

t_union_a(3行) + t_union_b(3行) = 共 **6行**（包含重复的Charlie记录）。

练习3 — UNION结果与区别：

UNION去重后共 **5行**（去除了重复的 (3, 'Charlie')）。

区别：

- UNION ALL：合并所有行，包含重复行，性能更高
- UNION（默认DISTINCT）：合并后去除完全重复的行，需要额外的去重操作

- 当不需要去重时，推荐使用UNION ALL（性能更好）

练习4 — UNION ALL行数：

California(5行) + Texas(5行) = 共 **10行**

练习5 — FROM子查询结果：

```
SELECT num
FROM (
    SELECT num, name FROM student
) tmp
WHERE tmp.name LIKE '王%';
```

姓王的同学学号：95003(王敏)、95014(王小丽)、95015(王君)、95018(王一)、95017(王风娟)

练习6 — 各州确诊数最多的州：

确诊总数超过100000的州（max_cases > 100000）：

- California（Los Angeles: 1,162,766）
- New York（New York City: 514,561）
- Texas（Bexar: 118,094）

练习7 — 不相关子查询（IN）结果：

employee的id：1201, 1202, 1203, 1204, 1206

employee_address的id：1201, 1202, 1204, 1206, 1207

共同存在（IN结果）：**id = 1201, 1202, 1204, 1206**，共4行记录。

练习8 — NOT IN结果：

employee_address中id不在employee中的记录：**id = 1207**（员工地址中存在此条记录，但employee表中无对应员工）。

练习9 — EXISTS相关子查询结果:

> 在employee_connection中有联系方式记录的员工:
> - 1201 (gopal)
> - 1203 (khalil)
> - 1204 (prasanth)
> - 1206 (kranthi)
>
> 共4名员工（1202 manisha在employee_connection中无记录）。

第八部分：CTE 公用表表达式 — 答案

练习1 — 高死亡人数加州县:

```
WITH high_deaths AS (
    SELECT county, cases, deaths
    FROM t_usa_covid19_p
    WHERE count_date = "2021-01-28"
    AND state = "California"
    AND deaths > 1000
)
SELECT * FROM high_deaths ORDER BY deaths DESC;
```

> 符合条件（deaths > 1000）的加州县:
> - Los Angeles（17096）
> - San Bernardino（2734）
> - Riverside（2997）
> - Orange（2538）
>
> 共4个县（根据实际数据）。

练习2 — 总死亡人数超过5000的州:

符合条件（total_deaths > 5000）的州：

- New York（约38336）
- California（约28705）
- Texas（约10130）
- Illinois（约7930）

练习3 — 同时满足高确诊和高死亡的县：

- Los Angeles, California（cases > 100000 且 deaths > 2000）
- New York City, New York（cases > 100000 且 deaths > 2000）

等（根据实际数据阈值而定）

练习4（扩展）— CTE改写：

```
WITH dept_stats AS (
    SELECT dept, count(*) AS stu_count, avg(age) AS avg_age
    FROM student
    WHERE dept IN ('CS', 'IS')
    GROUP BY dept
)
SELECT dept, stu_count, round(avg_age, 2) AS avg_age
FROM dept_stats
WHERE avg_age > 19 AND stu_count > 3;
```

第九部分：JOIN 连接操作 — 答案

三张表ID分析：

- employee 的 id：1201, 1202, 1203, 1204, 1206
- employee_address 的 id：1201, 1202, 1204, 1206, 1207
- employee_connection 的 id：1201, 1203, 1204, 1205, 1206
- 三表共同拥有的 id：**1201, 1204, 1206**

9.1 INNER JOIN

练习1：

> 结果共 **4行**，出现的员工：
> - 1201 gopal（jubilee）
> - 1202 manisha（ny）
> - 1204 prasanth（hyd）
> - 1206 kranthi（la）
>
> （1203 khalil在employee_address中无记录，1207在employee_address中有记录但employee中无对应员工）

练习2 — 三表INNER JOIN：

> 结果共 **3行**：
> - 1201 gopal（jubilee, 2356742, gopal@tp.com）
> - 1204 prasanth（hyd, 8887776, khalil@ac.com）
> - 1206 kranthi（la, 1231231, kranthi@tp.com）

9.2 LEFT JOIN

练习3：

> 结果共 **5行**（employee的所有5名员工都出现）：
> - 1201 gopal → phno: 2356742, email: gopal@tp.com
> - 1202 manisha → phno: **NULL**, email: **NULL**（无联系方式）
> - 1203 khalil → phno: 1661663, email: manisha@tp.com
> - 1204 prasanth → phno: 8887776, email: khalil@ac.com
> - 1206 kranthi → phno: 1231231, email: kranthi@tp.com
>
> **联系方式显示NULL的员工**：1202 manisha（employee_connection中无id=1202的记录）

9.3 RIGHT JOIN

练习5：

> 以employee_connection为右表（5行）全部返回：
> - id=1201 gopal → 正常显示
> - id=1203 khalil → 正常显示
> - id=1204 prasanth → 正常显示
> - id=1205 → e.id=**NULL**, e.name=**NULL**（employee中无id=1205的员工）
> - id=1206 kranthi → 正常显示
>
> **联系方式无对应员工的记录**：id=1205的联系方式（employee中无此员工）

RIGHT JOIN改写为LEFT JOIN：

```
SELECT e.id, e.name, e_conn.phno, e_conn.email
FROM employee_connection e_conn
LEFT JOIN employee e
ON e_conn.id = e.id;
```

9.4 FULL OUTER JOIN

练习6：

> 结果共 **6行**：
> - 1201 gopal - jubilee ✅
> - 1202 manisha - ny ✅
> - 1203 khalil - **NULL** - **NULL**（employee有，employee_address无）
> - 1204 prasanth - hyd ✅
> - 1206 kranthi - la ✅
> - **NULL** - **NULL** - hitec（employee_address有id=1207，employee无）
>
> **NULL值出现在**：khalil所在行（无住址）和1207所在行（无对应员工）

练习7 — JOIN结果行数对比：

JOIN类型	左表行数	右表行数	结果行数
INNER JOIN	5	5	**4**（共同id：1201,1202,1204,1206）
LEFT JOIN	5	5	**5**（以左表为准，右表不匹配显示NULL）
RIGHT JOIN	5	5	**5**（以右表为准，左表不匹配显示NULL）
FULL OUTER JOIN	5	5	**6**（两表所有行合并，1203和1207各贡献一行NULL）

9.5 LEFT SEMI JOIN

练习8：

```
SELECT * FROM employee e
LEFT SEMI JOIN employee_address e_addr
ON e.id = e_addr.id;
```

结果共 **4行**（employee中id在employee_address中存在的员工）：

- 1201 gopal
- 1202 manisha
- 1204 prasanth
- 1206 kranthi

与INNER JOIN的不同：LEFT SEMI JOIN只返回左表（employee）的列，不返回右表（employee_address）的city、street等字段。

练习9 — 关键区别：

> LEFT SEMI JOIN 与 INNER JOIN 的关键区别：
>
> 1. **返回列**：LEFT SEMI JOIN只返回左表列；INNER JOIN返回两表的列
> 2. **重复行**：若右表中有多行匹配，INNER JOIN会产生多条记录；LEFT SEMI JOIN左表每行最多返回一次（去重）
> 3. **性能**：LEFT SEMI JOIN通常比INNER JOIN效率更高（找到匹配即停止）

9.6 CROSS JOIN

练习10：

> employee(5行) × employee_address(5行) = **25行**（笛卡尔积）

练习11：

> 两种方式结果相同，都返回4行（id匹配的记录）。
>
> `CROSS JOIN + WHERE a.id = b.id` 等价于 `INNER JOIN ON a.id = b.id`，但CROSS JOIN先产生笛卡尔积再过滤，效率低于INNER JOIN（INNER JOIN在连接时即进行过滤）。

第十部分：综合挑战练习 — 参考答案

挑战1：

```
SELECT
    state,
    count(county)       AS county_count,
    sum(cases)          AS total_cases,
    sum(deaths)         AS total_deaths,
    round(avg(cases), 2) AS avg_cases
FROM t_usa_covid19_p
WHERE count_date = "2021-01-28"
GROUP BY state
```

```
ORDER BY total_cases DESC;
```

> 确诊数最多的3个州（参考结果）：
>
> 1. California（约1,879,984）
> 2. New York（约908,875）
> 3. Texas（约763,655）

挑战2：

```
WITH state_avg AS (
    SELECT state, avg(cases) AS avg_cases
    FROM t_usa_covid19_p
    WHERE count_date = "2021-01-28"
    GROUP BY state
)
SELECT t.county, t.state, t.cases, sa.avg_cases
FROM t_usa_covid19_p t
INNER JOIN state_avg sa ON t.state = sa.state
WHERE t.count_date = "2021-01-28"
AND t.cases > sa.avg_cases
ORDER BY t.state, t.cases DESC;
```

挑战3-①：

```
SELECT dept, sex, count(*) AS stu_count, round(avg(age), 2) AS avg_age
FROM student
GROUP BY dept, sex
ORDER BY dept, sex;
```

挑战3-②：

```
-- 方法1：使用子查询
SELECT dept, count(*) AS stu_count
FROM student
GROUP BY dept
HAVING count(*) = (
    SELECT max(cnt) FROM (
```

```
        SELECT dept, count(*) AS cnt
        FROM student GROUP BY dept
    ) t
);


-- 方法2：使用ORDER BY + LIMIT
SELECT dept, count(*) AS stu_count
FROM student
GROUP BY dept
ORDER BY stu_count DESC
LIMIT 1;
```

> 人数最多的系：CS（计算机系，共9名学生）

挑战3-③：

```
SELECT * FROM student
WHERE dept != 'CS'
AND age > (
    SELECT avg(age) FROM student WHERE dept = 'CS'
);
```

挑战4：

```
SELECT
    e.id,
    e.name,
    e.deg,
    e.salary,
    COALESCE(e_a.city, '未知')     AS address_city,
    COALESCE(e_conn.email, '未留') AS contact_email
FROM employee e
LEFT JOIN employee_address e_a ON e.id = e_a.id
LEFT JOIN employee_connection e_conn ON e.id = e_conn.id;
```

> `COALESCE(value, default)` 函数：如果value为NULL，则返回default值。

挑战5：

```
WITH ny_data AS (
    SELECT
        county,
        cases,
        deaths,
        round(deaths * 100.0 / cases, 4) AS death_rate
    FROM t_usa_covid19_p
    WHERE count_date = "2021-01-28"
    AND state = "New York"
    AND cases > 0
)
SELECT county, cases, deaths, death_rate
FROM ny_data
ORDER BY death_rate DESC
LIMIT 3;
```

> New York州死亡率最高的3个县通常是纽约市周边的较小县（死亡率 = deaths/cases × 100%）。

常见错误汇总与指导

错误类型	错误示例	正确写法	说明
WHERE中用聚合函数	`WHERE count(*) > 3`	`HAVING count(*) > 3`	聚合函数需在分组后使用
GROUP BY字段不在SELECT	`SELECT deaths FROM ... GROUP BY state`	`SELECT sum(deaths) FROM ... GROUP BY state`	非分组字段必须经过聚合函数
CLUSTER BY与ORDER BY混用	`CLUSTER BY sex ORDER BY age`	`DISTRIBUTE BY sex SORT BY age DESC`	CLUSTER BY不能与ORDER BY/SORT BY共用
LIMIT偏移量误解	`LIMIT 1, 5`（误以为从第1行取5条）	实际是从第2行开始取5条	LIMIT的偏移量从0开始
LEFT SEMI JOIN后引	`SELECT e_addr.city FROM e LEFT SEMI`	不能在SELECT中引用	LEFT SEMI JOIN只能

用右表列	`JOIN e_addr ON ...`	RIGHT表	返回左表列
UNION列数不一致	两个SELECT返回列数不同	确保两个SELECT列数和数据类型一致	UNION要求结构完全匹配

评分参考标准

部分	满分	说明
第一部分（环境准备）	10分	环境搭建成功且所有表数据加载正确
第二部分（SELECT基础）	10分	练习完成度
第三部分（WHERE与分区）	15分	重点考察WHERE限制理解
第四部分（GROUP BY/HAVING）	15分	重点考察WHERE与HAVING区别
第五部分（LIMIT/执行顺序）	10分	执行顺序填空题各2分
第六部分（高阶排序）	10分	CLUSTER BY与DISTRIBUTE BY理解
第七部分（UNION/子查询）	10分	子查询理解深度
第八部分（CTE）	10分	CTE语法正确性
第九部分（JOIN）	15分	六种JOIN类型的区别理解
第十部分（综合挑战）	15分	综合运用能力
合计	**120分**	**超出部分作为加分**

《Hive数据仓库应用教程》— 项目六综合实践工作手册 教师答案版