이 SQL 쿼리들은 다양한 조인(Join) 및 서브쿼리(Subquery)를 사용하여 여러 테이블 간의 데이터를 연결하고 추출하는 예시들입니다. 각각의 쿼리가 수행하는 작업을 하나씩 설명하겠습니다.

1. 사원(employee) 테이블과 부서(department) 테이블을 조인하여 부서번호, 부서명, 사원명, 사원번호를 출력 (조건: 사원명 '홍길동')

```sql
SELECT d.dept_id, d.dept_name, e.name, e.dept_id
FROM department d
CROSS JOIN employee e
WHERE e.name = '홍길동';
```

- **설명:** `department` 테이블과 `employee` 테이블을 **교차 조인(CROSS JOIN)** 을 사용하여 연결합니다. 결과적으로 두 테이블의 모든 가능한 조합이 생성됩니다. 하지만 `WHERE` 절에서 사원명이 '홍길동'인 경우로 필터링하여 출력합니다.

```sql
SELECT d.dept_id, d.dept_name, e.name, e.dept_id
FROM department d, employee e
WHERE e.name = '홍길동';
```

- **설명:** 동일한 쿼리지만 이번에는 **암시적 조인(Implicit Join)** 을 사용하여 동일한 결과를 얻습니다. 이 방법은 JOIN 키워드를 명시적으로 사용하지 않고, 단순히 쉼표로 테이블을 구분합니다.

2. 사원(employee) 테이블과 부서(department) 테이블을 조인하여 사원번호, 직위, 부서아이디, 부서명을 출력 (조건: 사원명 '홍길동')

```sql
SELECT e.emp_id, e.position, e.dept_id, d.dept_name
FROM employee e
INNER JOIN department d ON e.dept_id = d.dept_id
WHERE e.name = '홍길동';
```

- **설명:** `INNER JOIN`을 사용하여 `employee`와 `department` 테이블을 `dept_id` 필드를 기준으로 조인합니다. 이 조인은 두 테이블의 `dept_id`가 일치하는 행만 반환합니다. 결과는 홍길동이라는 사원의 부서 정보를 포함합니다.

```sql
SELECT e.emp_id, e.position, e.dept_id, d.dept_name
FROM employee e, department d
WHERE e.dept_id = d.dept_id AND e.name = '홍길동';
```

- **설명:** 동일한 조인 조건을 사용하되, 암시적 조인 방법을 사용하여 같은 결과를 얻습니다.

3. 고객 테이블(customer)과 주문 테이블(orders)을 조인하여 고객명, 담당자명, 회사명, 주문건수를 출력 (주문건수가 많은 순으로 정렬)

```
SELECT c.cust_id, c.contact_name, c.company_name, COUNT(*) AS order_count
FROM customer c
INNER JOIN orders o ON c.cust_id = o.cust_id
GROUP BY c.cust_id, c.contact_name, c.company_name
ORDER BY COUNT(*) DESC;
```

- **설명:** `INNER JOIN`을 통해 `customer` 테이블과 `orders` 테이블을 `cust_id` 필드를 기준으로 조인합니다. 각 고객의 주문 건수를 그룹화(`GROUP BY`)하고, 주문건수(`order_count`)가 많은 순서로 정렬(`ORDER BY COUNT(*) DESC`)합니다.

```
SELECT c.cust_id, c.contact_name, c.company_name, COUNT(*) AS order_count
FROM customer c, orders o
WHERE c.cust_id = o.cust_id
GROUP BY c.cust_id, c.contact_name, c.company_name
ORDER BY COUNT(*) DESC;
```

- **설명:** 동일한 조인과 그룹화를 수행하되, 암시적 조인 방법을 사용합니다.

4. 고객 테이블(customer)과 주문 테이블(orders) 및 주문 상세 테이블(order_details)을 조인하여 고객 아이디, 담당자명, 회사명, 주문 총액을 출력 (주문 총액 순으로 정렬)

```
SELECT c.cust_id, c.contact_name, c.company_name, SUM(od.quantity *
od.unit_price) AS total_order_amount
FROM customer c
INNER JOIN orders o ON c.cust_id = o.cust_id
INNER JOIN order_details od ON o.order_id = od.order_id
GROUP BY c.cust_id, c.contact_name, c.company_name
ORDER BY total_order_amount DESC;
```

- **설명:** 이 쿼리는 고객(`customer`), 주문(`orders`), 주문 상세(`order_details`) 세 테이블을 조인하여, 각 고객의 주문 총액을 계산합니다. `SUM(od.quantity * od.unit_price)`는 각 주문의 총 금액을 계산하며, 이를 기준으로 정렬합니다.

```
SELECT c.cust_id, c.contact_name, c.company_name, SUM(od.quantity *
od.unit_price) AS total_order_amount
FROM customer c, orders o, order_details od
WHERE c.cust_id = o.cust_id AND o.order_id = od.order_id
```

```
GROUP BY c.cust_id, c.contact_name, c.company_name
ORDER BY total_order_amount DESC;
```

- **설명:** 동일한 작업을 수행하지만, 암시적 조인을 사용하여 각 테이블을 연결합니다.

5. 고객 테이블(customer)과 마일리지 등급(mileage_grade) 테이블을 조인하여 고객 아이디, 회사명, 담당자명, 마일리지, 등급을 출력 (조건: 담당자명 '남성우')

```
SELECT c.cust_id, c.company_name, c.contact_name, c.mileage, mg.grade
FROM customer c
INNER JOIN mileage_grade mg ON c.mileage BETWEEN mg.min_mileage AND
mg.max_mileage
WHERE c.contact_name = '남성우';
```

- **설명:** 고객의 마일리지가 특정 범위에 속할 때 해당하는 마일리지 등급을 찾기 위해 `BETWEEN` 절을 사용하여 `customer`와 `mileage_grade` 테이블을 조인합니다. 필터링된 결과는 담당자명이 '남성우'인 고객만 출력됩니다.

```
SELECT c.cust_id, c.company_name, c.contact_name, c.mileage, mg.grade
FROM customer c, mileage_grade mg
WHERE c.mileage BETWEEN mg.min_mileage AND mg.max_mileage AND
c.contact_name = '남성우';
```

- **설명:** 동일한 조인 조건을 적용하지만, 암시적 조인 문법을 사용합니다.

6. 사원(employee) 테이블과 부서(department) 테이블을 외부 조인(LEFT OUTER JOIN)하여 사원번호, 사원명, 부서명을 출력 (부서가 없으면 NULL로 출력)

```
SELECT e.emp_id, e.name, d.dept_name
FROM employee e
LEFT OUTER JOIN department d ON e.dept_id = d.dept_id
WHERE e.gender = '여';
```

- **설명:** 이 쿼리는 `LEFT OUTER JOIN`을 사용하여 `employee` 테이블과 `department` 테이블을 조인합니다. `LEFT OUTER JOIN`은 왼쪽 테이블(`employee`)의 모든 행을 포함하며, 오른쪽 테이블(`department`)에 해당하는 데이터가 없는 경우 NULL 값을 반환합니다. 결과는 성별이 '여'인 사원만 출력합니다.

```sql
SELECT e.name, d.*
FROM employee e
LEFT OUTER JOIN department d ON e.dept_id = d.dept_id
WHERE d.dept_id IS NULL;
```

- **설명:** 이 쿼리는 부서 정보가 없는 사원들을 출력합니다. `LEFT OUTER JOIN`을 사용하여 부서 정보가 NULL인 행을 필터링합니다.

7. 사원(employee) 테이블을 조인하여 사원 아이디, 사원명, 부서장 아이디, 부서장명을 출력

```sql
SELECT e.emp_id, e.name, supervisor.emp_id AS "Supervisor Employee ID",
supervisor.name AS "Supervisor Name"
FROM employee e
INNER JOIN employee AS supervisor ON e.manager_no = supervisor.emp_id;
```

- **설명:** 이 쿼리는 `employee` 테이블을 스스로 조인하여 각 사원과 그들의 상사를 연결합니다. `INNER JOIN`을 사용하여 사원의 `manager_no`와 상사의 `emp_id`를 매칭합니다.

```sql
SELECT e.emp_id, e.name, supervisor.emp_id AS "Supervisor Employee ID",
supervisor.name AS "Supervisor Name"
FROM employee e, employee AS supervisor
WHERE e.manager_no = supervisor.emp_id;
```

- **설명:** 동일한 작업을 수행하되, 암시적 조인을 사용합니다.

8. 사원(employee) 테이블을 조인하여 사원 아이디, 사원명, 부서장 아이디, 부서장명을 출력 (부서장이 없으면 NULL 출력)

```sql
SELECT e.name AS name, e.position, supervisor.name AS supervisor_name
FROM employee AS supervisor
RIGHT OUTER JOIN employee e ON e.manager_no = supervisor.emp_id
ORDER BY supervisor_name;
```

- **설명:** `RIGHT OUTER JOIN`을 사용하여 모든 사원을 포함시키고, 해당 사원의 부서장이 없는 경우 NULL 값을 출력합니다.

9. 고객 테이블(customer)에서 최대 마일

리지를 가진 고객의 고객아이디, 회사명, 담당자명, 마일리지 출력

```sql
SELECT cust_id, company_name, contact_name, mileage
FROM customer
WHERE mileage = (SELECT MAX(mileage) FROM customer);
```

- **설명:** 이 쿼리는 서브쿼리를 사용하여 최대 마일리지를 가진 고객을 필터링합니다. `WHERE` 절의 서브쿼리는 전체 `customer` 테이블에서 가장 높은 마일리지 값을 찾습니다.

10. 고객 테이블(customer)에서 주문번호가 13번인 고객의 정보를 출력

```sql
SELECT company_name, contact_name
FROM customer
WHERE cust_id = (SELECT cust_id FROM orders WHERE order_id = 13);
```

- **설명:** 서브쿼리를 사용하여 주문번호가 13번인 고객의 `cust_id`를 가져와 해당 고객의 정보를 출력합니다.

```sql
SELECT c.company_name, c.contact_name
FROM customer c
INNER JOIN orders o ON c.cust_id = o.cust_id
WHERE o.order_id = 13;
```

- **설명:** `INNER JOIN`을 사용하여 주문번호가 13번인 고객의 정보를 출력합니다.

11. 고객 테이블(customer)에서 도시가 부산인 고객 중에서 최소 마일리지보다 큰 고객 정보를 출력

```sql
SELECT contact_name, company_name, mileage
FROM customer
WHERE mileage > (SELECT MIN(mileage) FROM customer WHERE city LIKE '%부
산%');
```

- **설명:** 이 쿼리는 서브쿼리를 사용하여 부산에 거주하는 고객 중 최소 마일리지를 찾고, 그보다 높은 마일리지를 가진 고객들을 필터링하여 출력합니다.

12. 고객 테이블(customer)에서 도시가 부산인 고객의 주문 건수를 주문 테이블(orders)에서 구하기

```sql
SELECT COUNT(*) AS order_count
FROM orders
```

```
WHERE cust_id IN (SELECT cust_id FROM customer WHERE city LIKE '%부산%');
```

- **설명:** 서브쿼리를 사용하여 부산에 거주하는 고객들의 `cust_id`를 찾고, 그 고객들의 주문 건수를 계산하여 출력합니다.

13. 고객 테이블(customer)에서 도시가 부산인 마일리지보다 큰 모든 고객의 정보를 출력

```
SELECT contact_name, company_name, mileage
FROM customer
WHERE mileage > ANY (SELECT mileage FROM customer WHERE city LIKE '%부
산%');
```

- **설명:** `ANY` 키워드를 사용하여 부산에 거주하는 고객들 중 하나라도 마일리지가 큰 고객들을 출력합니다.

```
SELECT contact_name, company_name, mileage
FROM customer
WHERE mileage > ALL (SELECT AVG(mileage) FROM customer GROUP BY region);
```

- **설명:** `ALL` 키워드를 사용하여 각 지역별 평균 마일리지보다 높은 마일리지를 가진 고객들을 출력합니다.

14. 주문 테이블과 고객 테이블에서 주문이 있는 고객의 정보를 출력

```
SELECT cust_id, company_name
FROM customer
WHERE EXISTS (SELECT 1 FROM orders WHERE cust_id = customer.cust_id);
```

- **설명:** `EXISTS` 키워드를 사용하여 `orders` 테이블에 주문이 있는 고객의 정보를 필터링하여 출력합니다.

```
SELECT cust_id, company_name
FROM customer
WHERE cust_id IN (SELECT DISTINCT cust_id FROM orders);
```

- **설명:** `IN` 서브쿼리를 사용하여 주문이 있는 고객의 `cust_id`를 필터링하고, 고객 정보를 출력합니다.

```
SELECT DISTINCT c.cust_id, c.company_name
FROM customer c
```

```
INNER JOIN orders o ON c.cust_id = o.cust_id;
```

- **설명:** `INNER JOIN`을 사용하여 주문이 있는 고객만을 출력합니다. `DISTINCT`는 중복된 고객 정보를 제거합니다.

15. 고객 테이블(customer)에서 도시별 평균 마일리지를 구하되, 고객의 평균 마일리지보다 큰 경우만 출력

```
SELECT city, AVG(mileage) AS avg_mileage
FROM customer
GROUP BY city
HAVING AVG(mileage) > (SELECT AVG(mileage) FROM customer);
```

- **설명:** `GROUP BY`와 `HAVING` 절을 사용하여 도시별 평균 마일리지를 계산하고, 전체 고객 평균 마일리지보다 높은 도시만 출력합니다.

16. 고객 테이블(customer)에서 도시별 평균 마일리지와 고객의 평균 마일리지 차이를 구하기

```
SELECT c.contact_name, c.company_name, c.mileage, c.city,
city_avg.avg_mileage, city_avg.avg_mileage - c.mileage AS difference
FROM customer c
INNER JOIN (
    SELECT city, AVG(mileage) AS avg_mileage
    FROM customer
    GROUP BY city
) city_avg ON c.city = city_avg.city;
```

- **설명:** 서브쿼리를 사용하여 도시별 평균 마일리지를 계산하고, 이 값을 각 고객의 마일리지와 비교하여 차이를 계산합니다.

```
WITH city_summary AS (
    SELECT city, AVG(mileage) AS avg_mileage
    FROM customer
    GROUP BY city
)
SELECT c.contact_name, c.company_name, c.mileage, c.city, cs.avg_mileage,
cs.avg_mileage - c.mileage AS difference
FROM customer c
INNER JOIN city_summary cs ON c.city = cs.city;
```

- **설명:** `WITH` 절을 사용하여 도시별 평균 마일리지 서브쿼리를 정의하고, 이를 사용하여 각 고객의 마일리지 차이를 계산합니다.

17. 고객 테이블(customer)과 주문 테이블(orders)에서 고객별로 가장 최근 주문일자를 구하기

```
SELECT cust_id, contact_name, (SELECT MAX(order_date) FROM orders WHERE
orders.cust_id = customer.cust_id) AS last_order_date
FROM customer;
```

- **설명:** 서브쿼리를 사용하여 각 고객별로 가장 최근의 주문 일자를 계산하여 출력합니다.

18. 사원 테이블(employee)에서 사원번호, 사원명, 매니저 번호, 매니저 이름을 출력

```
SELECT emp_id, name, manager_no, (SELECT name FROM employee AS supervisor
WHERE supervisor.emp_id = e.manager_no) AS supervisor_name
FROM employee e;
```

- **설명:** 서브쿼리를 사용하여 각 사원의 상사 이름을 찾고, 이를 출력합니다.

19. 고객 테이블(customer)에서 도시 별 최대 마일리지를 가진 고객 정보를 출력

```
SELECT city, contact_name, company_name, mileage
FROM customer
WHERE (city, mileage) IN (SELECT city, MAX(mileage) FROM customer GROUP BY
city);
```

- **설명:** 복합 조건 `(city, mileage)`를 사용하여 각 도시별로 최대 마일리지를 가진 고객의 정보를 출력합니다.

이 쿼리들은 여러 가지 복잡한 데이터를 조인하고 필터링하는 방법을 보여주며, 각 쿼리의 핵심은 SQL 조인의 이해와 서브쿼리를 효과적으로 사용하는 것입니다.