utrozet : indexing & SQL

INDEX

Index:

Index in the data structured technique that helps to retrive data quickly from database.

Syntax:

Create indexing index_name
ON table_name (column 1, column 2, ...);

Type of indexing

- ordered indices
  ↳ Base of sorted ordering values
- Hash indices
  ↳ base on values determine by function called hash function

Structure:

Search Key	Data Reference pointer

↳ Primary candidate key sorted order

↳ Pointer holding the address of the disk block.

S.K	D.R
1	1001
2	1004
3	1006
4	1008

indexing methods → Ordered, Primary, clustering, Secondary

Primary → Dense index, sparse index

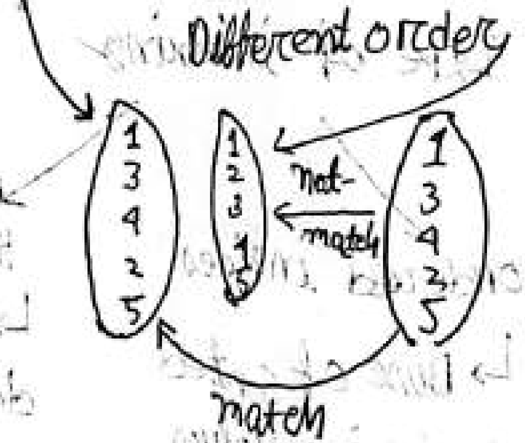
i) **Ordered indices:**

In an ordered index, index entries are stored on the search key value

Primary/clustering index

In search key of the index has some order as the sequential order of the file.

Secondary/Non-clustering index



ii) **clustered indexing:**

In this two or more columns are grouped together to uniquely identify the records.

↳ Records with similar characteristics are grouped together and indexes are created for this groups.]

Ex.

Dept_id	ename
1	A
1	B
2	C
2	D
3	E
4	F

Dept_id	pointer
1	1000 → (1 A, 1 B)
2	1010 → (2 C, 2 D)
3	1020
4	1030

Primary index:

Data is stored according to the search key (Primary key to table)

↳ allowes Sequential file organization.

↳ Primary key is used to create index.

↳ Efficient

Types:-

Dense Index

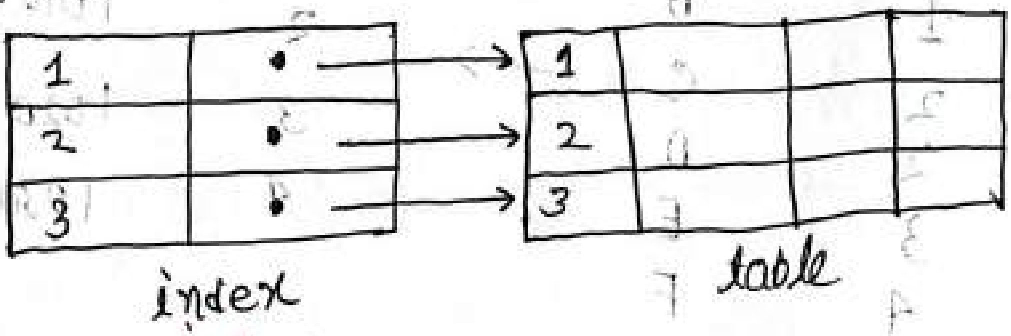
Index record for every search key value.

Sparse Index

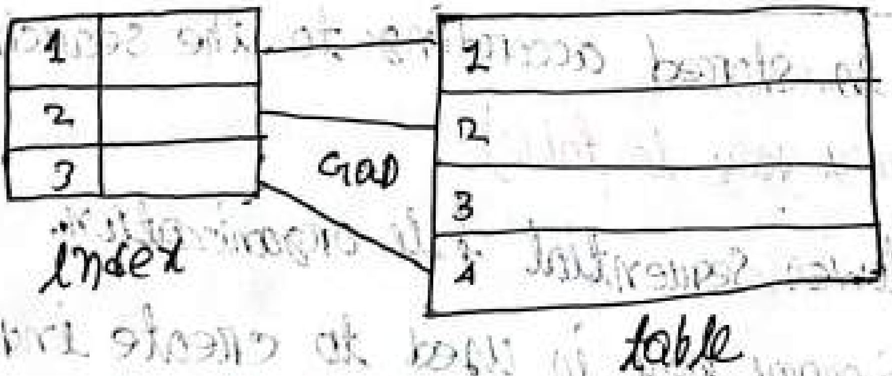
Index record for only few item. Each item points to Block.

Dense index:

No. of records is same as no. of indexes
↳ in table



Sparse index:



Secondary index:

It is used to optimize query processing and access records in database with some information other than primary key.

Levels used:

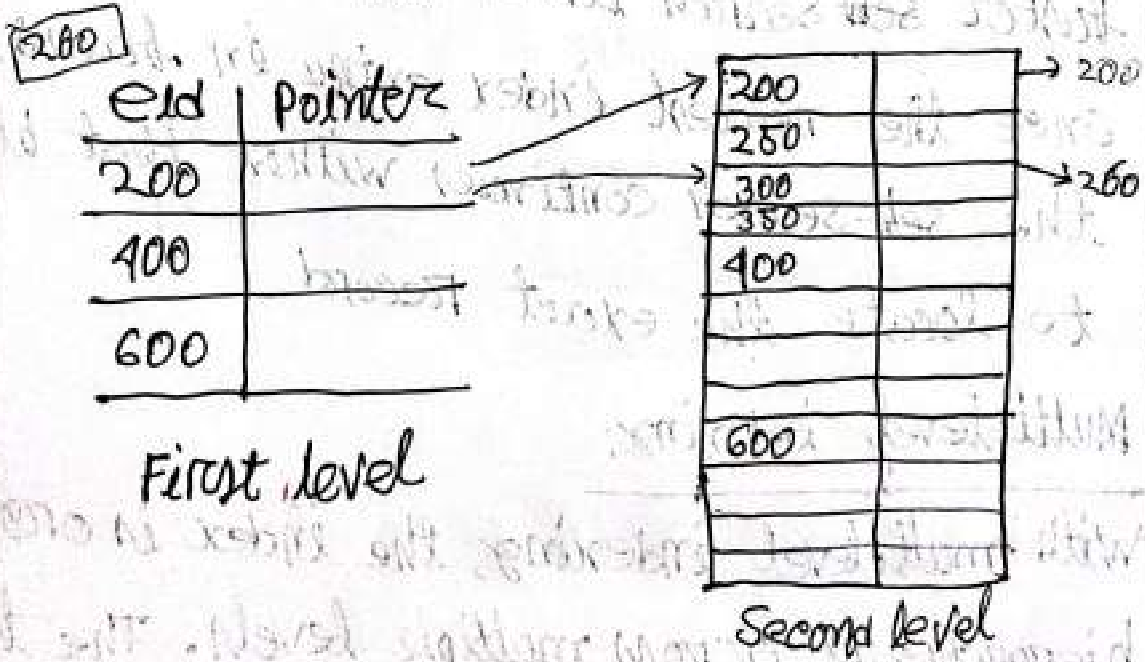
First Level

Large range of no. is selected.

Second Level

Actual physical locⁿ of data

eid	pointer
1	→
2	→
500	→
2000	→



Question 20-21

Topic: indexing

8.b) What is sparse indexing? How does multilevel indexing improve the efficiency of searching an Index file?

Solve:

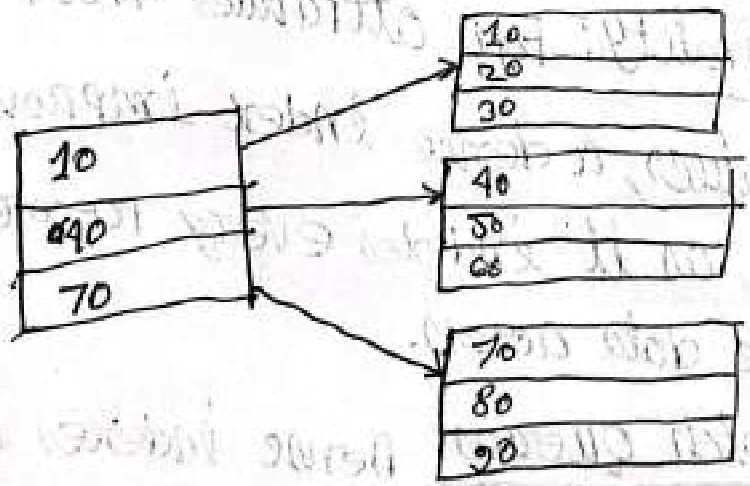
Sparse Indexing:

In sparse indexing, only certain records are indexed - usually the first record of each block. This allows a smaller index file and faster ~~sch~~ search times within the index. Once the nearest index entry is found, the ~~sch~~ search continues within that block to locate the exact record.

~~Multi~~

Multilevel indexing:

Multilevel indexing improves efficiency by creating a hierarchical structure of indexes, where the first level indexes point to second-level indexes and so on. This reduces the number of disk accesses needed to locate data, as the search progresses down the level of indexes rather than scanning a single large index. The hierarchical structure allows faster data retrieval especially for large databases.



c) When is it preferable to use a dense index rather than a sparse index? Explain your answer

Solve:

A dense index is preferable to a sparse index in the following scenarios:

1. Frequent Searches: If the application performs many searches, a dense index allows for quicker lookups since it contains an entry for every search key, enabling direct access to records

2. High cardinality: For attributes with many unique values, a dense index improves retrieval efficiency, as it indexes every record, allowing for precise data access.

3. Exact Match Queries: Dense indexes are optimal for ~~eax~~ exact match queries because they provide direct access to all records, enhancing performance in these case.

8d) Now make necessary modification to the index file after deletion of the record for the account no 'A-5' and then 'A-2'

Index file after delation of the record for the account no 'A-5':

Branch name	Pointer
Adabor	→ A-9
Dhanmondi	→ A-8
Mirpur	→ A-2
MotiJheel	→ A-6

Index file after deletion of the record for the account no 'A-2'

Branch-name	Pointer
Adabor	→ A-9
Dhanmondi	→ A-8
Mirpur	→ A-4
Motijheel	→ A-6

Q - 2019-20

8 (a) Here's a brief differentiation for each type of indexing:

1. Primary indexing: An index base on a table's unique primary key; it organizes records in order and allows fast access.

2. Secondary indexing:

An index on non-primary, non-unique fields; allows multiple indexes per table for quicker lookups on specific columns.

3. Clustering indexing:

Build on non-unique columns with repeated values; groups similar records together, which is efficient for range searches.

b) 2021-21(c)

c) In general, it is not possible to have two primary indices on the same relation for different keys because the tuples in a relation would have to be stored in different order to have same values stored together. We could accom-plish this by storing the relation twice and duplicating all values, but for a centralized system, this is not efficient.

d) 1) Instance of Relation:

Course relation:

Course-name	room	instructor
Math 101	R_1	Prof. Smith
Physics 101	R_2	Prof. Johnson

enrollment Relation:

Courses_name	Student-name	grade
Math101	Alice	A
Math 101	Bob	B
Math 101	Carol	C
Physics 101	Dave	B
Physics 101	Eve	A
physics 101	Frank	C

Clustering structure:

Data is clustered based on courses-name storing each courses and its corresponding enrollment records together:

- cluster 1 (Math 101): (Math 101, R1, Prof. smith), (Math 101, Bob, B), (Math 101, Carol, C)
- cluster 2 (Physics 101): (physics 101, R_2, Prof. Johnson), (physics101, Dave, B), (Physics101, Eve, A), (Physics101, Frank, C).

2020 - 21

a) Consider the database schema below: [10]

employee (ename, street, city)

emp_company (ename, cname, salary, jdate)

company (cname, city)

manager (ename, mname, shift)

Note: A manager is also an employee of a company.

Give SQL and RA expressions for the following queries:

→ R.A not needed this is out of syllabus.

(i) Find names, street addresses and cities of residence of all employees who work under manager Sabbir and who joined before January 01, 2019.

(ii) Find the names of the employees living in the same city where Rahim is residing.

(iii) Display the average salary of each company except Square Pharma.

(iv) Increase the salary of employees by 10% for the companies those are located in Barisal.

(v) Delete records from emp_company that contain employees living in Rajshahi.

b) SQL allows a foreign-key dependency to refer to the same relation, as in the following example: [2]

```
CREATE TABLE manager
    (employee-name  CHAR(20),
    manager-name  CHAR(20),
    PRIMARY KEY employee-name,
    FOREIGN  KEY  (manager-name)  REFERENCES  manager(employee-
    name) ON DELETE CASCADE);
```

Here, *employee-name* is a key to the table *manager*, meaning that each employee has at most one manager. The foreign-key clause requires that every manager also be an employee. Explain exactly what happens when a tuple in the relation *manager* is deleted.

(i) Find names, street addresses, and cities of residence of all employees who work under manager Sabbir and who joined before January 01, 2019.

SQL Query:

Sql

```sql
SELECT e.ename, e.street, e.city
FROM employee e
JOIN emp_company ec ON e.ename = ec.ename
JOIN manager m ON e.ename = m.ename
WHERE m.mname = 'Sabbir' AND ec.jdate < '2019-01-01';
```

Relational Algebra:

$$\pi_{ename,street,city}\left(\sigma_{mname='Sabbir' \wedge jdate<'2019-01-01'}(employee \bowtie emp\_company \bowtie manager)\right)$$

(ii) Find the names of the employees living in the same city where Rahim is residing.

SQL Query:

Sql   Copy code

```sql
SELECT e1.ename
FROM employee e1
JOIN employee e2 ON e1.city = e2.city
WHERE e2.ename = 'Rahim'; [redacted]
```

Relational Algebra:

$$\pi_{e1.ename}\left(\sigma_{e1.city=e2.city \land e2.ename='Rahim' \land e1.ename \neq 'Rahim'}(employee \times employee)\right)$$

(iii) Display the average salary of each company except Square Pharma.

SQL:

Sql   Copy code

```sql
SELECT cname, AVG(salary) AS average_salary
FROM emp_company
WHERE cname != 'Square Pharma'
GROUP BY cname;
```

Relational Algebra:

$$\gamma_{cname, AVG(salary)}\left(\sigma_{cname \neq 'SquarePharma'}(emp\_company)\right)$$

(iv) Increase the salary of employees by 10% for the companies that are located in Barisal.

SQL:

Sql   Copy code

```sql
UPDATE emp_company
SET salary = salary * 1.1
WHERE cname IN (SELECT cname FROM company WHERE city = 'Barisal');
```

Relational Algebra:

Relational Algebra does not have an update operation, so it is not expressible in traditional RA.

(v) Delete records from emp_company that contain employees living in Rajshahi.

SQL:

Sql   Copy code

```sql
DELETE FROM emp_company
WHERE ename IN (SELECT ename FROM employee WHERE city = 'Rajshahi');
```

b) When a tuple (row) in the manager table is deleted, ON Delete cascade triggers an automatic deletion of any rows where that tuple's employee_name is referenced as manager-name. This deletion process continues recursively, ensuring that any employees managed directly or indirectly by the deleted employee are also removed, preserving referential integrity within the table

Shortest:

When a row in the manager table is deleted, ON DELETE CASCADE automatically deletes all rows where that employee is listed as a manager, continuing recursively until no references remain.

5. a) Consider the database schema below: [8]

worker (<u>wname</u>, street, city)
works (<u>work_id</u>, wname, orgname, salary, jdate)
organization (<u>orgname</u>, city)
manages (<u>wname</u>, manager-name, shift)

Note: A manager is also an employee of an organization.
Give SQL expressions for the following queries:

(i) Find the names of all employees who work for "Google".
(ii) Find the names of all employees in this database who live in the same city as the company for which they work.
(iii) Find the names of all employees who live in the same city and on the same street as do their managers.
(iv) Give all managers in the database a 7.5% salary raise.
(v) Find the names of the employees living in the same city where Rahim is residing.
(vi) Find the company with the most employees
(vii) Create a view to show all the employees who earn more than average salary.
(viii) Find all the employees who work more than five years for "Facebook".

(i) Find the names of all employees who work for "Google".

```sql
SELECT wname
FROM works
WHERE orgname = 'Google';
```

(ii) Find the names of all employees who live in the same city as the company for which they work.

```sql
SELECT w.wname
FROM worker w
JOIN works ws ON w.wname = ws.wname
JOIN organization o ON ws.orgname = o.orgname
WHERE w.city = o.city;
```

(iii) Find the names of all employees who live in the same city and on the same street as their managers.

```sql
SELECT w.wname
FROM worker w
JOIN manages m ON w.wname = m.wname
JOIN worker wm ON m.manager-name = wm.wname
WHERE w.city = wm.city AND w.street = wm.street;
```

(iv) Give all managers in the database a 7.5% salary raise.

```sql
UPDATE works
SET salary = salary * 1.075
WHERE wname IN (SELECT wname FROM manages);
```

(v) Find the names of the employees living in the same city where Rahim is residing.

Sql Copy code

```sql
SELECT wname
FROM worker
WHERE city = (SELECT city FROM worker WHERE wname = 'Rahim');
```

(vi) Find the company with the most employees.

Sql Copy code

```sql
SELECT orgname
FROM works
GROUP BY orgname
ORDER BY COUNT(wname) DESC
LIMIT 1;
```

(vii) Create a view to show all the employees who earn more than the average salary.

Sql Copy code

```sql
CREATE VIEW AboveAverageSalary AS
SELECT wname, salary
FROM works
WHERE salary > (SELECT AVG(salary) FROM works);
```

(viii) Find all the employees who have worked for "Facebook" for more than five years.

Sql Copy code

```sql
SELECT wname
FROM works
WHERE orgname = 'Facebook' AND DATEDIFF(CURDATE(), jdate) > 5 * 365;
```

b) SQL allows a foreign-key dependency to refer to the same relation, as in the following [4] example:

```
CREATE TABLE manager(
employee-name  CHAR(20),
manager-name  CHAR(20),
PRIMARY KEY employee-name,
FOREIGN  KEY  (manager-name)  REFERENCES  manager(employee-
name) ON DELETE CASCADE);
```

Here, *employee-name* is a key to the table *manager*, meaning that each employee has at most one manager. The foreign-key clause requires that every manager also be an employee.

(i) Explain exactly what happens when a tuple in the relation *manager* is deleted.

(ii) What will happen, if ***RESTRICT*** is used instead of ***CASCADE***?

(i) When a tuple (row) in the `manager` table is deleted with `ON DELETE CASCADE`, any rows that reference the deleted `employee-name` as their `manager-name` will also be deleted automatically. This deletion will continue recursively, ensuring that all employees who directly or indirectly report to the deleted employee are also removed, preserving referential integrity.

(ii) If `RESTRICT` is used instead of `CASCADE`, deletion of a tuple is restricted if it has dependent rows (i.e., if other rows reference its `employee-name` as their `manager-name`). In this case, attempting to delete such a row will produce an error, preventing the deletion until all dependent rows are removed or updated to maintain referential integrity.

5. a) Consider the following table and give an expression in SQL for each of the following queries: [10]

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manages(employee_name, manager_name)

(i) Find the names of all employees who work for First Bank Corporation.

(ii) Find all employees in the database who live in the same cities as the companies for which they work.

(iii) Find all employees in the database who live in the same cities and on the same streets as do their managers.

(iv) Find all employees who earn more than the average salary of all employees of their company.

(v) Find the company that has the smallest payroll.

(vi) Give all employees of First Bank Corporation a 10 percent raise.

(vii) Find the company that has the most employees.

(viii) Find the employees who earn highest salary.

(ix) Find all employees who earn more than the average salary of all.

(x) Create a new table '*employer*' with the attributes *employer_id, employee_name, company_name*, where primary key is (*employer_id, company_name*) and foreign key is *employee_name*.

b) Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$, with primary keys A, C, and E respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $(r_1 \bowtie r_2 \bowtie r_3)$ and give an efficient strategy for computing the join. [2]

(i) Find the names of all employees who work for First Bank Corporation.

Sql

```sql
SELECT employee_name
FROM works
WHERE company_name = 'First Bank Corporation';
```

(ii) Find all employees in the database who live in the same cities as the companies for which they work.

Sql

```sql
SELECT e.employee_name
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
JOIN company c ON w.company_name = c.company_name
WHERE e.city = c.city;
```

(iii) Find all employees in the database who live in the same cities and on the same streets as do their managers.

Sql

```sql
SELECT e.employee_name
FROM employee e
JOIN manages m ON e.employee_name = m.employee_name
JOIN employee em ON m.manager_name = em.employee_name
WHERE e.city = em.city AND e.street = em.street;
```

(iv) Find all employees who earn more than the average salary of all employees of their company.

Sql

```sql
SELECT w.employee_name
FROM works w
JOIN (
    SELECT company_name, AVG(salary) AS avg_salary
    FROM works
    GROUP BY company_name
) avg_salary ON w.company_name = avg_salary.company_name
WHERE w.salary > avg_salary.avg_salary;
```

(v) Find the company that has the smallest payroll.

Sql

```sql
SELECT company_name
FROM works
GROUP BY company_name
ORDER BY SUM(salary) ASC
LIMIT 1;
```

(vi) Give all employees of First Bank Corporation a 10 percent raise.

Sql Copy code

```sql
UPDATE works
SET salary = salary * 1.10
WHERE company_name = 'First Bank Corporation';
```

(vii) Find the company that has the most employees.

Sql Copy code

```sql
SELECT company_name
FROM works
GROUP BY company_name
ORDER BY COUNT(employee_name) DESC
LIMIT 1;
```

(viii) Find the employees who earn the highest salary.

Sql Copy code

```sql
SELECT employee_name
FROM works
WHERE salary = (SELECT MAX(salary) FROM works);
```

(ix) Find all employees who earn more than the average salary of all.

Sql Copy code

```sql
SELECT employee_name
FROM works
WHERE salary > (SELECT AVG(salary) FROM works);
```

(x) Create a new table `employer` with the attributes `employer_id`, `employee_name`, and `company_name`, where the primary key is `(employer_id, company_name)` and `employee_name` is a foreign key referencing `employee`.

Sql Copy code

```sql
CREATE TABLE employer (
    employer_id INT,
    employee_name CHAR(20),
    company_name CHAR(20),
    PRIMARY KEY (employer_id, company_name),
    FOREIGN KEY (employee_name) REFERENCES employee(employee_name)
);
```

b)

'To efficiently estimate and compute the join size of :

1. Estimate Size:

First, join with on . This join yields 1500 tuples (since has 1500 tuples).

Then, join the result with on , yielding a maximum of 1000 tuples (matching the primary key size of).

Estimated size of : 1000 tuples.

2. Efficient Strategy:

Perform first to minimize intermediate results, then join the result with .

5. a) Consider the following relations: [10]

client (client-no, name, address, city)

product (product-no, description, profit-percent, qty-in-hand, reorder-level, cost-price)

salesman (salesman-no, name, address, city, sale-amt)

salesorder (order-no, order-date, client-no, del-add, salesman-no, del-date, order-status)

order-detail (order-no, product-no, qty-ordered, qty-delivered)

Give SQL and RA expressions for the following queries:

(i) Find the list of all clients who stay in cities Dhaka or Khulna.

(ii) Find the products with their description whose selling price is greater than 2000 and less than or equal to 5000. [**Hints**: Selling price can be found from cost-price and profit-percent]

(iii) Find the total ordered and delivered quantity for each product with a product range of P0035 to P0056.

(iv) Find the clients with their names and order numbers whose orders are handled by the salesman Mr. X.

(v) Find the product no and description of non-moving products, i.e., products not being sold.

→ don't know

b) Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$, with primary keys A, C, and E respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $(r_1 \bowtie r_2 \bowtie r_3)$ and give an efficient strategy for computing the join [2]

(i) List of all clients who stay in cities "Dhaka" or "Khulna"

SQL

Sql Copy code

```sql
SELECT name, city
FROM client
WHERE city IN ('Dhaka', 'Khulna');
```

Relational Algebra

$$\pi_{\text{name, city}}(\sigma_{\text{city}='Dhaka' \text{ OR city}='Khulna'}(\text{client}))$$

(ii) Find products with their descriptions whose selling price is greater than 2000 and less than or equal to 5000

Selling price is calculated as:

$$\text{selling price} = \text{cost-price} + (\text{cost-price} \times \frac{\text{profit-percent}}{100})$$

SQL

Sql Copy code

```sql
SELECT product-no, description
FROM product
WHERE (cost-price + (cost-price * profit-percent / 100)) >
2000
  AND (cost-price + (cost-price * profit-percent / 100)) <=
5000;
```

Relational Algebra

Letting
$\text{selling price} = \text{cost-price} + (\text{cost-price} \times \frac{\text{profit-percent}}{100})$:

$$\pi_{\text{product-no, description}}(\sigma_{2000<\text{selling price}\leq 5000}(\text{product}))$$

(iii) Total ordered and delivered quantity for each product with product numbers between "P0035" and "P00S6"

SQL

Sql Copy code

```sql
SELECT product-no,
       SUM(qty-ordered) AS total_ordered,
       SUM(qty-delivered) AS total_delivered
FROM order-detail
WHERE product-no BETWEEN 'P0035' AND 'P00S6'
GROUP BY product-no;
```

Relational Algebra

$$\gamma_{\text{product-no,SUM(qty-ordered),SUM(qty-delivered)}}(\sigma_{\text{product-no}\geq'P0035' \text{ AND product-no}\leq'P00S6'}(\text{order-detail})$$

(iv) Clients with their names and order numbers whose orders are handled by the salesman "Mr. X"

SQL

Sql Copy code

```sql
SELECT c.name, s.order-no
FROM client c
JOIN salesorder s ON c.client-no = s.client-no
JOIN salesman sm ON s.salesman-no = sm.salesman-no
WHERE sm.name = 'Mr. X';
```

Relational Algebra

$$\pi_{\text{c.name, s.order-no}}(\sigma_{\text{sm.name}='Mr.X'}(\text{client} \bowtie \text{salesorder} \bowtie \text{salesman})$$

(v) Product number and description of non-moving products (products not being sold)

SQL

Sql Copy code

```sql
SELECT p.product-no, p.description
FROM product p
LEFT JOIN order-detail od ON p.product-no = od.product-no
WHERE od.product-no IS NULL;
```

Relational Algebra

$$\pi_{\text{product-no, description}}(\text{product} \setminus (\pi_{\text{product-no}}(\text{order-detail})))$$