香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF
DATA SCIENCE
數據科學學院

Ack: Prof. Jignesh Patel @ CMU
Dr. Chui Chun Kit @ HKU

CSC3170
2: SQL *part a*

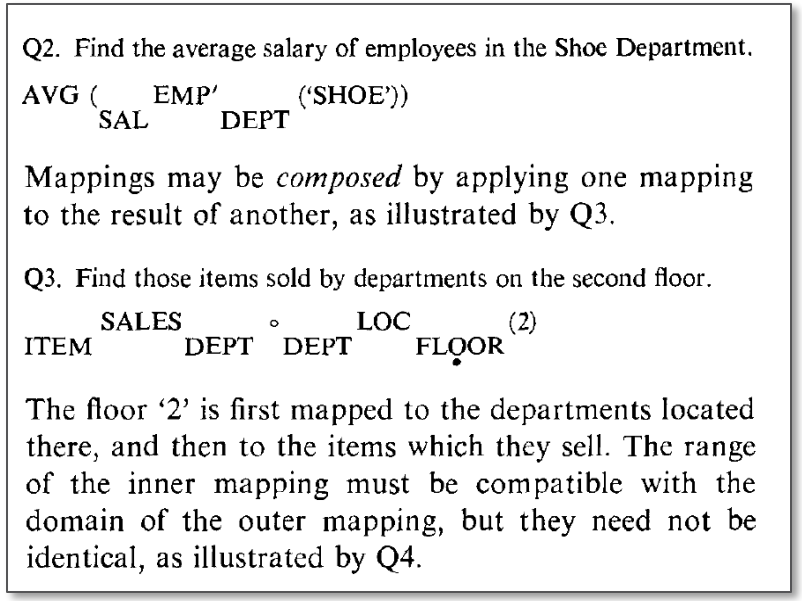
Chenhao Ma
School of Data Science
The Chinese University of Hong Kong, Shenzhen

Last Lecture

- We introduced the Relational Model as the superior data model for databases.
- We then showed how Relational Algebra is the building blocks that will allow us to query and modify a relational database.

SQL History

- In 1971, IBM created its first relational query language called SQUARE.
- IBM then created "SEQUEL" in 1972 for IBM System R prototype DBMS.
  - Structured English Query Language
- IBM releases commercial SQL-based DBMSs:
  - System/38 (1979), SQL/DS (1981), and DB2 (1983).



Q2. Find the average salary of employees in the Shoe Department.

$$\text{AVG}\left(\underset{\text{SAL}}{\text{EMP}'} \underset{\text{DEPT}}{} (\text{'SHOE'})\right)$$

Mappings may be *composed* by applying one mapping to the result of another, as illustrated by Q3.

Q3. Find those items sold by departments on the second floor.

$$\underset{\text{ITEM}}{\text{SALES}} \underset{\text{DEPT}}{} \circ \underset{\text{DEPT}}{\text{LOC}} \underset{\text{FLOOR}}{} (2)$$

The floor '2' is first mapped to the departments located there, and then to the items which they sell. The range of the inner mapping must be compatible with the domain of the outer mapping, but they need not be identical, as illustrated by Q4.

SQL History

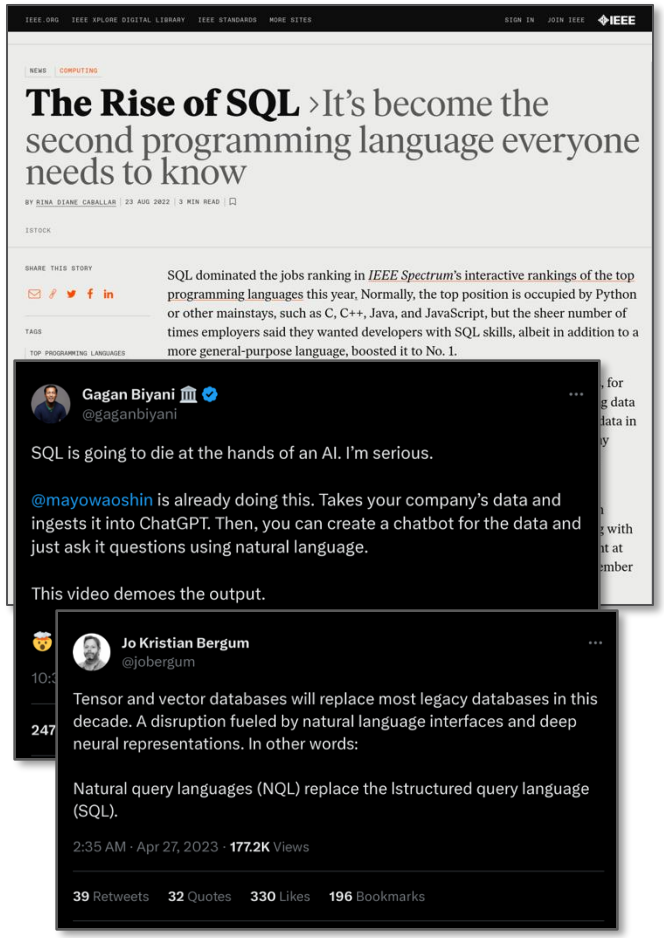
- ANSI Standard in 1986. ISO in 1987

→Structured Query Language

- Current standard is **SQL:2023**

→ **SQL:2023** → Property Graph Queries, Muti-Dim. Arrays
→ **SQL:2016** → JSON, Polymorphic tables
→ **SQL:2011** → Temporal DBs, Pipelined DML
→ **SQL:2008** → Truncation, Fancy Sorting
→ **SQL:2003** → XML, Windows, Sequences, Auto-Gen IDs.
→ **SQL:1999** → Regex, Triggers, OO

- The minimum language syntax a system needs to say that it supports SQL is SQL-92.



Structured Query Language (SQL)

SQL is the combination of

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Data Control Language (DCL)
- Transaction Control Language (TCL)

Important: SQL is based on **bags** (duplicates) not **sets** (no duplicates).

DDL: Data Definition

DDL: Create Table

- A database table is defined using the `CREATE TABLE` command
  - Create a table to store the bank branch info

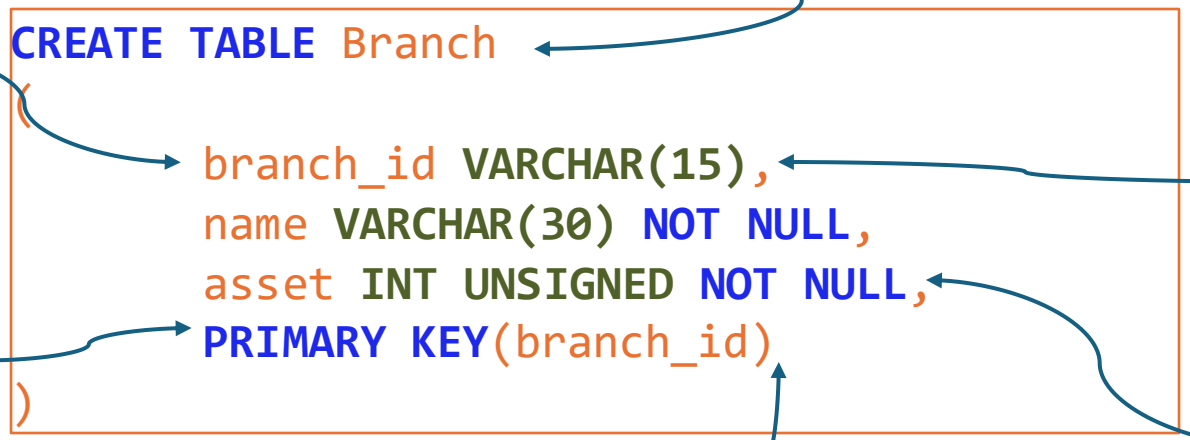


Table name (cannot be a keyword in database, e.g., **CREATE**)

Column name

Column type

Constraint (Primary Key), optional in this statement, can add back later using the **ALTER TABLE** command.

NOT NULL means each record's value in the column must not be a null value.

PRIMARY KEY automatically ensures **NOT NULL**.

No comma in the last instruction.

DDL: Drop Table

- `DROP TABLE` deletes all information about the dropped table from the database.

```
DROP TABLE Branch;
```

  - The DBMS may **reject** the `DROP TABLE` instruction when the table is referenced by another table via some constraints (e.g., **referential constraints**).

ArtistAlbum(artist_id, album_id)

artist_id	album_id
1	11
3	11
3	22
2	33

Artist(id, name, year, origin)

id	name	year	origin
1	The Chainsmokers	2012	US
2	Imagine Dragon	2008	US
3	Coldplay	1997	UK

After the foreign key is established, if we drop the `Artist` table, the records in the `ArtistAlbum` table will lost their references .

DDL: Alter Table

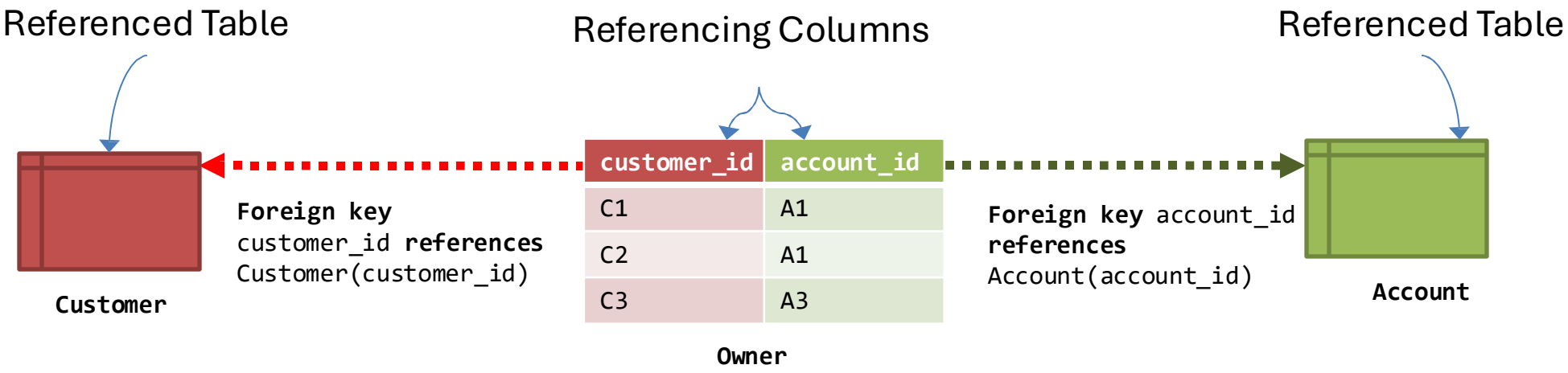
- `ALTER TABLE` can be used to
  - Add columns to an existing table.
  - `ALTER TABLE Branch ADD branch_phone INT (12);`

  - Remove a column from a table.
  - `ALTER TABLE Branch DROP branch_phone;`

  - Add constraints (e.g., PRIMARY KEY) to a table.
  - `ALTER TABLE Branch ADD PRIMARY KEY (branch_id);`

DDL: Foreign Key Constraints

- A **foreign key** is a referential constraint between two tables.
  - The columns in the referencing table must reference the columns of the **primary key** or other **superkey** in the referenced table.
  - I.e., The value in one row of the **referencing columns** must occur in a **single row** in the referenced table **(Why?).**



DDL: Foreign Key Constraints

- The foreign key can be established in the **CREATE TABLE** command.

```
CREATE TABLE Owner (
   customer_id VARCHAR(15),
   account_id VARCHAR(15),
   PRIMARY KEY(customer_id, account_id),
   FOREIGN KEY(customer_id) REFERENCES Customer(customer_id),
   FOREIGN KEY(account_id) REFERENCES Account(account_id)
);
```

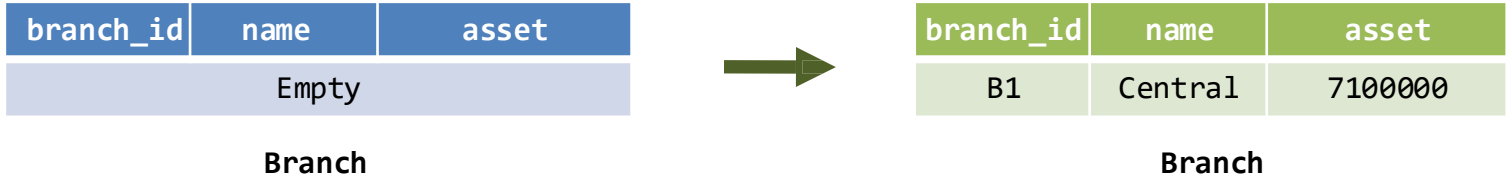
- The foreign key can also be defined using the **ALTER TABLE** command.

```
ALTER TABLE Owner
ADD FOREIGN KEY (customer_id) REFERENCES Customer(customer_id);
```

DML: Insert, Delete and Update

DML: Insert

- The **INSERT INTO** command is used to insert records (tuples) into the database table.




```
INSERT INTO Branch VALUES ( 'B1' , 'Central', 7100000);
```

Table name (case sensitive)

Value in the **1st** column

Value in the **2nd** column

Value in the **3rd** column

- Inserting multiple records

```
INSERT INTO Branch VALUES
( 'B2' , 'Causeway Bay', 9000000),
( 'B3' , 'Aberdeen', 400000);
```

DML: Insert

- Most DBMS provide an alternative way to insert large amount of records into a table.
  - `COPY` in PostgreSQL

```
COPY Branch FROM '/path/to/branch.csv' WITH
(FORMAT csv);
```

```
B1,Central,7100000
B2,Causeway Bay;9000000
B3;Aberdeen;400000
B4; North Point;3700000
…
```

branch.csv

DML: Delete

- The **`DELETE FROM`** command is used to delete records (tuples) from a database table.
  - **Example:** Delete all records from the Branch table.

branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	name	asset
Empty		

Branch

```
DELETE FROM Branch;
```

DML: Delete

- The **DELETE FROM** command is used to delete records (tuples) from a database table.
  - Example: Delete the branch "Central" from the Branch table.

branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	name	asset
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

```
DELETE FROM Branch WHERE name = 'Central';
```

The tuples that satisfy the conditions specified here are deleted.

DML: Update

- The **UPDATE** command is used to update records (tuples) from a database table.
  - Example: Update the asset of branch with branch_id 'B1' to $0.

branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	name	asset
B1	Central	0
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

```
UPDATE Branch
SET  asset = 0
WHERE branch_id = 'B1';
```

DML: Update

- The **UPDATE** command can also be used with **arithmetic expressions**.
  - Example: Increase all accounts with balances over $500 by 6%.

account_id	branch_id	balance
A1	B1	500
A2	B2	400
A3	B2	900
A4	B1	700

Account

account_id	branch_id	balance
A1	B1	500
A2	B2	400
A3	B2	954
A4	B1	742

Account

```
UPDATE Account
SET balance = balance * 1.06
WHERE balance > 500;
```

DML: Update

- The **UPDATE** command can also be used with **arithmetic expressions**.
  - Example: Increase all accounts with balances under $500 by 5% and all other accounts by 6%.

account_id	branch_id	balance
A1	B1	500
A2	B2	400
A3	B2	900
A4	B1	700

Account

account_id	branch_id	balance
A1	B1	530
A2	B2	420
A3	B2	954
A4	B1	742

Account

```
UPDATE Account
SET balance = balance * 1.05
WHERE balance < 500;
```

```
UPDATE Account
SET balance = balance * 1.06
WHERE balance >= 500;
```

The order of executing these two is important!
Which first?

DML: Update

- The **CASE** command can be used to perform conditional update.

```
UPDATE Account
SET balance = CASE
WHEN balance < 500 THEN balance *1.05
ELSE balance * 1.06
END
```

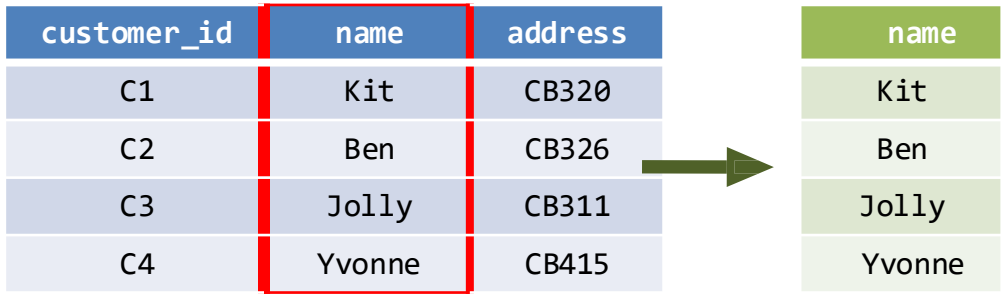
Note: When there are multiple WHEN … THEN in the query, only the first true statement (from top to bottom) will be executed.

DQL: Querying

Select

DQL: the SELECT clause

- The **SELECT** clause lists the attributes desired in the result of a query.
  - Example: Find the names of all customers.



customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Customer

name
Kit
Ben
Jolly
Yvonne

Result

```
SELECT name FROM Customer;
```

- An asterisk in the select clause denotes “all attributes”
  - Example: List all column values of all customer records.

```
SELECT * FROM Customer;
```

DQL: the SELECT clause

- The **SELECT** clause can contain **arithmetic expressions** (+, –, *, /) operating on constants or attributes of tuples.
  - Example: List the loan_id and amount of each loan record, display the amount in USD (originally stored in HKD).

loan_id	branch_id	amount
L1	B3	900
L2	B2	1500
L3	B1	1000

Loan

loan_id	amount /7.8
L1	115.385
L2	192.308
L3	128.205

Loan

```
SELECT loan_id, amount/7.8
FROM Loan;
```

DQL: the FROM clause

- The **FROM** clause lists the relations (tables) involved in the query.
  - Example: Find the **Cartesian product** of Customer and Borrower

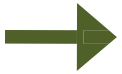
```
SELECT *
FROM Customer, Borrower;
```

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Customer

customer_id	loan_id
C1	L3
C4	L2
C2	L1

Borrower



Cartesian product of A and B means generate **all possible pairs** of records from A and B.

customer_id	name	address	customer_id	loan_id

DQL: the FROM clause

- The **FROM** clause lists the relations (tables) involved in the query.
  - Example: Find the **Cartesian product** of Customer and Borrower

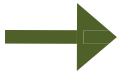
```
SELECT *
FROM Customer, Borrower;
```

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Customer

customer_id	loan_id
C1	L3
C4	L2
C2	L1

Borrower



Cartesian product of A and B means generate **all possible pairs** of records from A and B.

customer_id	name	address	customer_id	loan_id
C1	Kit	CB320	C1	L3

DQL: the FROM clause

- The **FROM** clause lists the relations (tables) involved in the query.
  - Example: Find the **Cartesian product** of Customer and Borrower

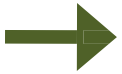
```
SELECT *
FROM Customer, Borrower;
```

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Customer

customer_id	loan_id
C1	L3
C4	L2
C2	L1

Borrower



Cartesian product of A and B means generate **all possible pairs** of records from A and B.

customer_id	name	address	customer_id	loan_id
C1	Kit	CB320	C1	L3
C2	Ben	CB326	C1	L3

DQL: the FROM clause

- The **FROM** clause lists the relations (tables) involved in the query.
  - Example: Find the **Cartesian product** of Customer and Borrower

```
SELECT *
FROM Customer, Borrower;
```

Cartesian product is the most primitive way of joining two tables. However, many resulting tuples are *not very useful*. Therefore, we often need to specify the **joining condition** to filter out the non-meaningful results.

customer_id	name	address	customer_id	loan_id
C1	Kit	CB320	C1	L3
C2	Ben	CB326	C1	L3
C3	Jolly	CB311	C1	L3
C4	Yvonne	CB415	C1	L3
C1	Kit	CB320	C4	L2
C2	Ben	CB326	C4	L2
C3	Jolly	CB311	C4	L2
C4	Yvonne	CB415	C4	L2
C1	Kit	CB320	C2	L1
C2	Ben	CB326	C2	L1
C3	Jolly	CB311	C2	L1
C4	Yvonne	CB415	C2	L1

DQL: the WHERE clause

- The **WHERE** clause specifies **conditions** that the result must satisfy.
  - Example: For each loan, find out the name of the customer who borrow the loan.

customer_id	loan_id
C1	L3
C4	L2
C2	L1

Borrower

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Customer

Step 1. What are the table(s) that contain the information to answer this query?

Observation 1.
First, the information of customers (customer_id) who borrow loan is in the **Borrower** table.

Observation 2.
Second, we need to find out the name of the customer, the name is in the **Customer** table.

DQL: the WHERE clause

```
SELECT Borrower.loan_id, Customer.name
FROM Customer, Borrower
```

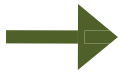
Step 2. Now we want to relate two tables, if no conditions is specified, **Cartesian product** will be returned. What is the joining condition?

customer_id	loan_id
C1	L3
C4	L2
C2	L1

Borrower

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Customer



customer_id	name	address	customer_id	loan_id
C1	Kit	CB320	C1	L3
C2	Ben	CB326	C1	L3
C3	Jolly	CB311	C1	L3
C4	Yvonne	CB415	C1	L3
C1	Kit	CB320	C4	L2
C2	Ben	CB326	C4	L2
C3	Jolly	CB311	C4	L2
C4	Yvonne	CB415	C4	L2
C1	Kit	CB320	C2	L1
C2	Ben	CB326	C2	L1
C3	Jolly	CB311	C2	L1
C4	Yvonne	CB415	C2	L1

DQL: the WHERE clause

```
SELECT Borrower.loan_id, Customer.name
FROM Customer, Borrower
WHERE Customer.customer_id =
Borrower.customer_id
```

loan_id	name
L3	Kit
L2	Yvonne
L1	Ben

Result

customer_id	loan_id
C1	L3
C4	L2
C2	L1

Borrower

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Customer

customer_id	name	address	customer_id	loan_id
C1	Kit	CB320	C1	L3
C2	Ben	CB326	C1	L3
C3	Jolly	CB311	C1	L3
C4	Yvonne	CB415	C1	L3
C1	Kit	CB320	C4	L2
C2	Ben	CB326	C4	L2
C3	Jolly	CB311	C4	L2
C4	Yvonne	CB415	C4	L2
C1	Kit	CB320	C2	L1
C2	Ben	CB326	C2	L1
C3	Jolly	CB311	C2	L1
C4	Yvonne	CB415	C2	L1

DQL: the WHERE clause

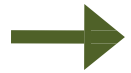
- The **WHERE** clause specifies **conditions** that the result must satisfy.
- Comparison results can be combined using logical connectives **AND**, **OR**, and **NOT**.
  - Example: Find all loan ID of loans made at branch_id B1 with loan amounts > $1200

There are two conditions in the query!

```
SELECT loan_id
FROM Loan
WHERE branch_id = 'B1' AND
      amount > 1200;
```

branch_id	loan_id	amount
B3	L1	900
B1	L2	1500
B1	L3	1000

Loan

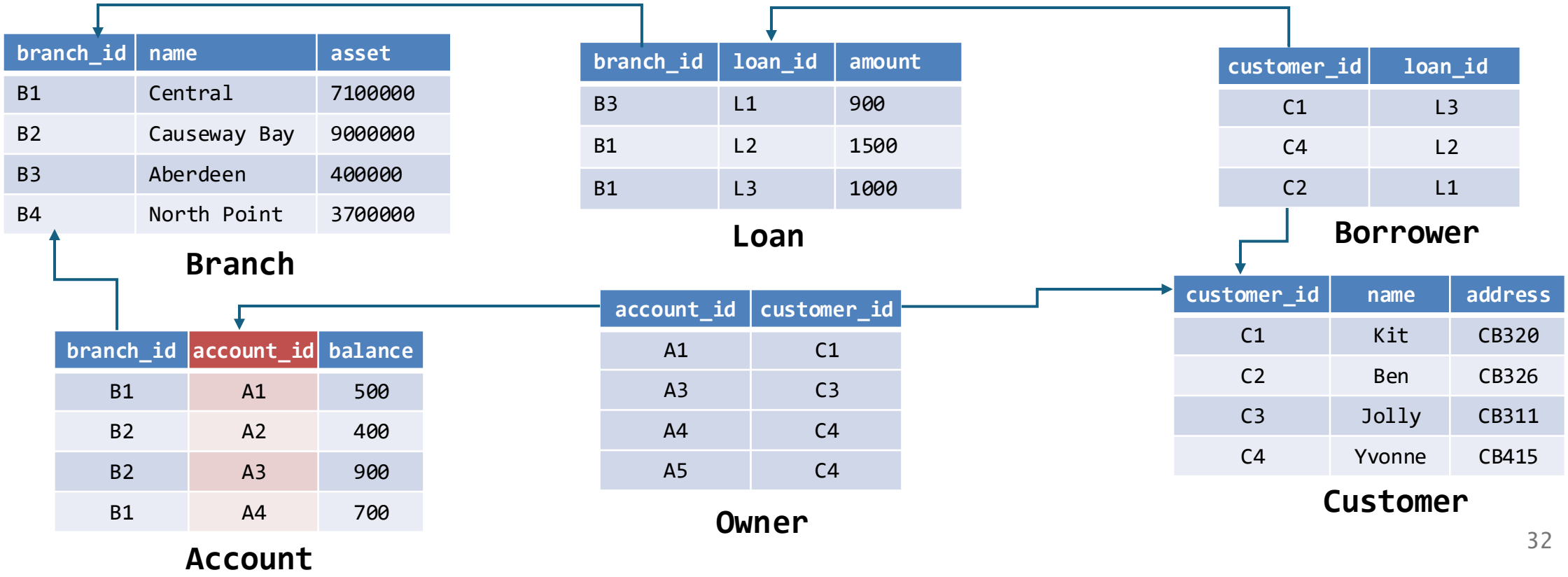


loan_id
L2

Result

Exercise

- **Query**: Find the names of all branches that have a loan.
  - **Step 1.** Identify the tables that contain the necessary information to answer the query.



branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	loan_id	amount
B3	L1	900
B1	L2	1500
B1	L3	1000

Loan

customer_id	loan_id
C1	L3
C4	L2
C2	L1

Borrower

branch_id	account_id	balance
B1	A1	500
B2	A2	400
B2	A3	900
B1	A4	700

Account

account_id	customer_id
A1	C1
A3	C3
A4	C4
A5	C4

Owner

customer_id	name	address
C1	Kit	CB320
C2	Ben	CB326
C3	Jolly	CB311
C4	Yvonne	CB415

Customer

Exercise

- **Query**: Find the names of all branches that have a loan.
  - **Step 1.** Identify the tables that contain the necessary information to answer the query.

branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	loan_id	amount
B3	L1	900
B1	L2	1500
B1	L3	1000

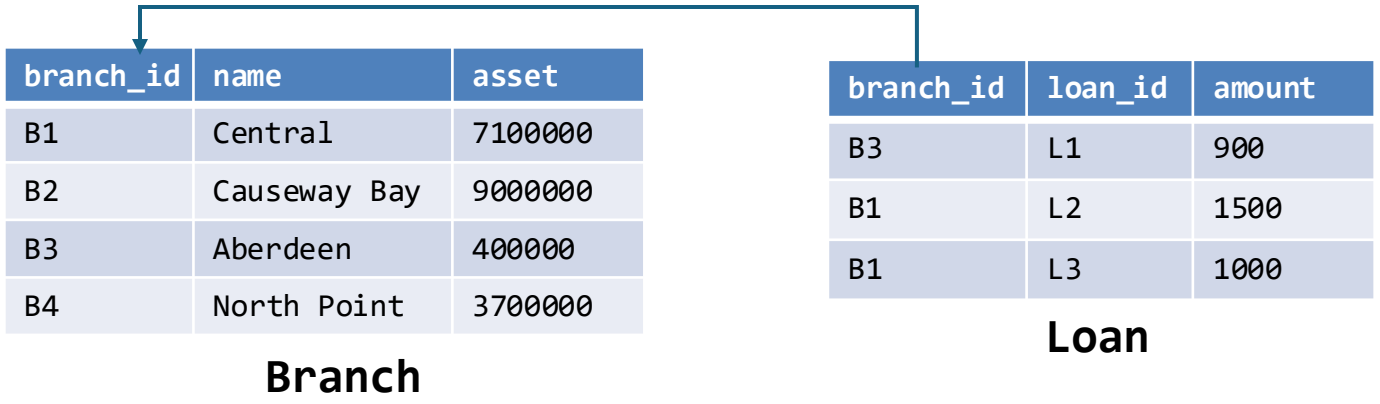
Loan

  - **Step 2.** Construct the `SELECT` statement.

```
SELECT ?
FROM Branch, Loan
WHERE ?
;
```

Exercise

- **Query**: Find the names of all branches that have a loan.
  - **Step 1.** Identify the tables that contain the necessary information to answer the query.



branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	loan_id	amount
B3	L1	900
B1	L2	1500
B1	L3	1000

Loan

  - **Step 2.** Construct the `SELECT` statement.

```
SELECT Branch.name
FROM Branch, Loan
WHERE ?
;
```

Exercise

- **Query**: Find the names of all branches that have a loan.
  - **Step 1.** Identify the tables that contain the necessary information to answer the query.

branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	loan_id	amount
B3	L1	900
B1	L2	1500
B1	L3	1000

Loan

  - **Step 2.** Construct the `SELECT` statement.

```
SELECT Branch.name
FROM Branch, Loan
WHERE Branch.branch_id = Loan.branch_id
;
```

Joining condition

Usually, when **linking the information of two tables**, we need to specify **the joining condition.** Often we need to join the columns that participate in the referential constraint between the two tables.

Exercise

- **Query**: Find the names of all branches that have a loan.
  - **Step 1.** Identify the tables that contain the necessary information to answer the query.

branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	loan_id	amount
B3	L1	900
B1	L2	1500
B1	L3	1000

Loan

Duplicate values returned!

  - **Step 2.** Construct the `SELECT` statement.

```
SELECT Branch.name
FROM Branch, Loan
WHERE Branch.branch_id = Loan.branch_id
;
```

name
Central
Central
Aberdeen

Exercise

- **Query**: Find the names of all branches that have a loan.
  - **Step 1.** Identify the tables that contain the necessary information to answer the query.

branch_id	name	asset
B1	Central	7100000
B2	Causeway Bay	9000000
B3	Aberdeen	400000
B4	North Point	3700000

Branch

branch_id	loan_id	amount
B3	L1	900
B1	L2	1500
B1	L3	1000

Loan

You can eliminate duplicate values in the results by using the **DISTINCT** keyword.

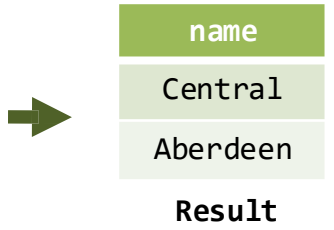
  - **Step 2.** Construct the **SELECT** statement.

```
SELECT DISTINCT Branch.name
FROM Branch, Loan
WHERE Branch.branch_id = Loan.branch_id
;
```

name
Central
Aberdeen

DQL: Renaming (SELECT AS)

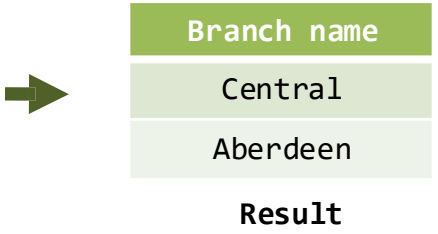
```
SELECT DISTINCT Branch.name
FROM Branch, Loan
WHERE Branch.branch_id = Loan.branch_id
;
```



- Rename can be operated on both tables and **attributes.**

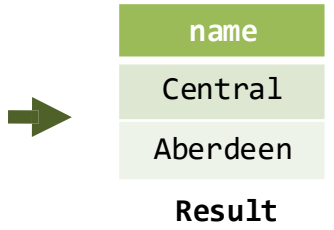
We use the keyword **AS** to signify renaming.

```
SELECT DISTINCT Branch.name AS 'Branch name'
FROM Branch, Loan
WHERE Branch.branch_id = Loan.branch_id
;
```



DQL: Renaming (SELECT AS)

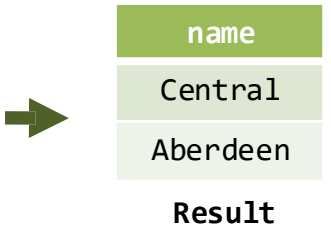
```
SELECT DISTINCT Branch.name
FROM Branch, Loan
WHERE Branch.branch_id = Loan.branch_id
;
```



- Rename can be operated on both **tables** and attributes.

The two SQLs are equivalent to each other.

```
SELECT DISTINCT B.name
FROM Branch B, Loan L
WHERE B.branch_id = L.branch_id
;
```



Conclusion

SQL

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)