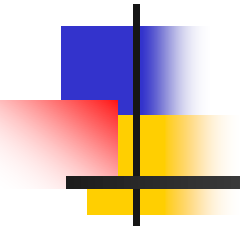


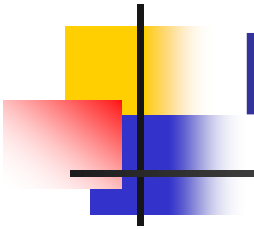
COMP1140: Database and Information Management

Lecture Note – Week 7

Dr Suhuai Luo

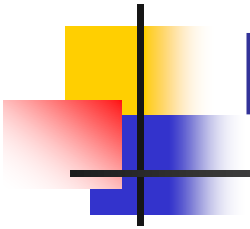
School of EEC

University of Newcastle



Notice

- Feedbacks/comments on Assignment 1
-
- Assignment 2 is due Friday next week (Sept 21) at 4pm.
  - In your A2, make sure work on a working EER, based on feedback from your marker, or the partial EER model given



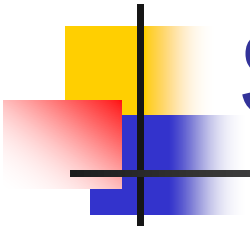
Last Week

- Relational algebra
  - Selection
  - Projection
  - Cartesian Product
  - Union
  - Set Difference
  - Intersection
  - Division
  - Joins (Theta, Equi, Natural, Outer)
  - Aggregate and Grouping
- Q?

This week

- Structured Query Language (SQL)
  - Historical Perspective
  - DDL
    - Integrity constraints and SQL
    - CREATE, ALTER and DROP
  - DML
    - INSERT
    - UPDATE
    - DELETE
    - SELECT Basics
- More on A2
- Reference: ch 6 & 7

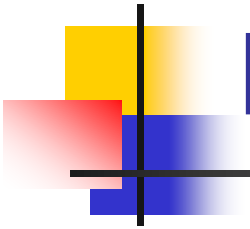
SQL: Historical Perspective



- In 1974, D. Chamberlin at IBM San Jose Laboratory defined language called ‘Structured English Query Language’ (SEQUEL)
- A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL
- IBM subsequently produced a prototype DBMS called *System R*, based on SEQUEL/2

SQL: Historical Perspective (contd.)

- Later, SQL was standardized:
  - In 1987, ANSI and ISO published an initial standard for SQL
  - In 1989, ISO published an addendum that defined an ‘Integrity Enhancement Feature’
  - In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92
  - In 1999, SQL:1999 was released with support for object-oriented data management
  - In late 2003, SQL:2003 was released.
  - In 2008, SQL:2008 was released

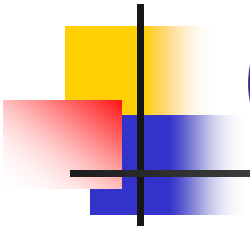


Popularity of SQL

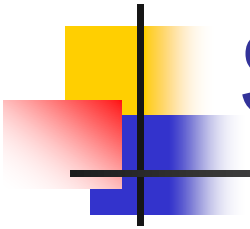
- SQL is the most popularly used database language today!

- ISO standards for SQL makes it both the formal and *de facto* standard language for relational databases.

Structured Query Language (SQL)



- Major advantage of standardization is that application programs are independent of the DBMS
  - Conformance to the standard means same standard SQL commands work on different DBMSs
- However, most vendors have DBMS specific extensions: dialects
  - E.g. Microsoft's Transact-SQL, Oracle's PL/SQL, ...

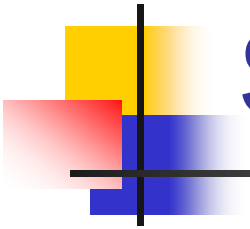


SQL (contd.)

SQL is a comprehensive database language:

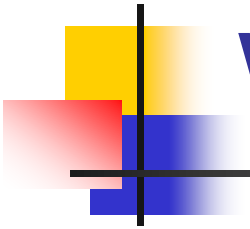
- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Facilities for security & authorization
- Facilities for transaction processing
- Facilities for embedding SQL in general purpose languages (Embedded SQL)
- ...

*We will learn a small but important subset of these!



SQL: Terminology

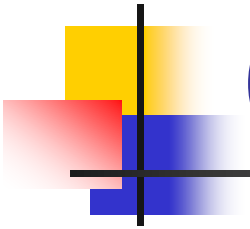
Relational Model	SQL
Relation	Table
Attribute	Column
Tuple	Row



Writing SQL commands

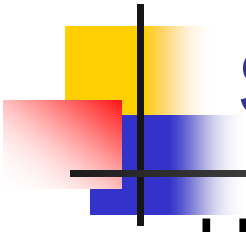
- SQL statement consists of *reserved words* and *user-defined words*
- Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines
- User-defined words are made up by user and represent names of various database objects such as relations, columns, views

Writing SQL commands (contd.)

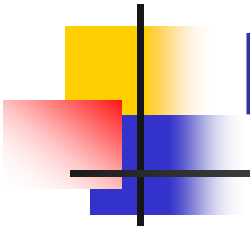


- Most components of an SQL statement are *case insensitive.*
- More readable with indentation and lineation:
  - Each clause better begin on a new line.
  - Start of a clause should line up with start of other clauses.
  - If clause has several parts, should each appear on a separate line and be indented under start of clause.

Notation used to describe syntax



- Use extended form of BNF notation to describe SQL syntax:
  - Upper-case letters represent reserved words
  - Lower-case letters represent user-defined words
  - | indicates a *choice* among alternatives
  - Curly braces {a} indicate a *required element*
  - Square brackets [a] indicate an *optional element*.
  - Ellipsis (...) indicates *optional repetition* (0 or more)
  - E.g., {a|b} (,c...)



Literals

- Literals are constants used in SQL statements. They must be typed exactly as they appear in DB.
- All non-numeric literals must be enclosed in single quotes (e.g. ‘Newcastle’).
- All numeric literals must not be enclosed in quotes (e.g. 650.00).

ISO SQL Data Types

Table 6.1 ISO SQL data types.

Data type	Declarations			
boolean	BOOLEAN			
character	CHAR	VARCHAR		
bit	BIT	BIT VARYING		
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
datetime	DATE	TIME	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

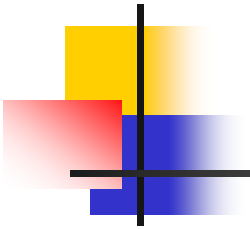
SQL's Data Definition Language (DDL)

- CREATE – Creates a database object
  - CREATE TABLE
  - CREATE VIEW
  - CREATE INDEX

  ...
- ALTER – Changes the schema and constraints of a database object
  - ALTER TABLE
  - ALTER VIEW

  ...
- DROP – Removes a database object
  - DROP TABLE
  - DROP VIEW

  ...



Integrity Constraints

- Five types of integrity constraints:
  - required data
  - domain constraints
  - entity integrity
  - referential integrity
  - general constraints

Integrity Constraints (contd.)

- Required Data:
  - enforced using NOT NULL
  - E.g. position VARCHAR(10) NOT NULL
- Default Data:
  - Default value when data is not specified
  - E.g. sex CHAR NOT NULL DEFAULT 'M'
- Domain Constraints:
  - Enforced by selecting a data type for domain of attribute
  - optionally creating a CHECK constraint or using CREATE DOMAIN
    - (a) CHECK
      E.g. sex CHAR NOT NULL DEFAULT 'M' CHECK (sex IN ('M', 'F'))

Integrity Constraints (contd.)

(b) CREATE DOMAIN:

Provides an name for a domain

CREATE DOMAIN DomainName [AS] dataType
[DEFAULT defaultOption]
[CHECK (searchCondition)]

For example:

```
CREATE DOMAIN SexType AS CHAR
        CHECK (VALUE IN ('M', 'F'));
sex     SexType NOT NULL
```

Integrity Constraints (contd.)

- Entity integrity constraints:
  - Primary key values are unique and does not contain NULL values
    - E.g. PRIMARY KEY(staffNo)

      PRIMARY KEY(clientNo, propertyNo)
  - Only one primary key per table
  - Alternate keys use UNIQUE constraints + NOT NULL
    - E.g.

      title CHAR(10) NOT NULL
      state CHAR(10) NOT NULL
      city CHAR(10) NOT NULL

      UNIQUE(title), UNIQUE(state), UNIQUE(city)

      UNIQUE(state, city)

Integrity Constraints (contd.)

- Referential integrity constraints:
  - Foreign Key (FK) is column or set of columns that links each row in child table containing FK to row of parent table containing matching Primary Key (PK)
  - Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table
  - Enforced using FOREIGN KEY()...REFERENCES

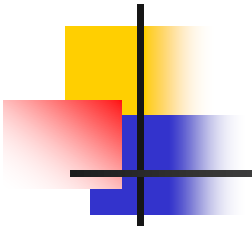
Example,

FOREIGN KEY(branchNo) REFERENCES Branch(branchNo)

Integrity Constraints (contd.)

- Maintaining referential integrity with updates and deletes:
  - Any INSERT/UPDATE attempting to create FK value in **child table** without matching PK value in parent is rejected
  - Action taken attempting to update/delete a PK value in **parent table** with matching rows in child is dependent on referential action specified:
    - NO ACTION
    - CASCADE
    - SET NULL
    - SET DEFAULT

Integrity Constraints (contd.)



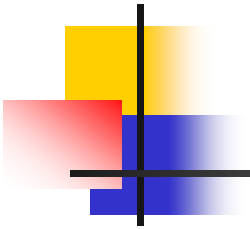
Data Manipulations causing Referential Actions:

- UPDATE (specified using ON UPDATE)
- DELETE (specified using ON DELETE)

Referential Actions (for ON DELETE):

- CASCADE: Delete matching rows in child, and so on in cascading manner
- SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns donot have NOT NULL specified.
- SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.
- NO ACTION: Reject delete from parent. This is the default.

- For ON UPDATE, similar actions to delete except update child rows rather than deleting



Integrity Constraints (contd.)

- Example:
  - FOREIGN KEY (ownerNo) REFERENCES Owner ON UPDATE CASCADE ON DELETE SET NULL

Integrity Constraints (contd.)

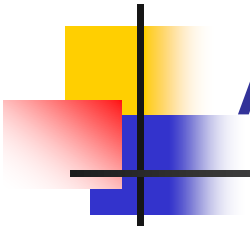
- General Constraints:
  - Use CHECK constraint - at the row level only, or
  - TRIGGER - across rows or multiple tables
  - Example

```
CONSTRAINT StaffNotHandlingTooMuch
CHECK (NOT EXISTS (SELECT staffNo
                   FROM PropertyForRent
                   GROUP BY staffNo
                   HAVING COUNT(*) > 100))
```

Example: CREATE TABLE

- CREATE TABLE example with constraints

```
CREATE TABLE Register (
  stdNo          CHAR(5),
  courseID       CHAR(8),
  semesterID     INTEGER REFERENCES Semester ON UPDATE CASCADE
                 ON DELETE NO ACTION,
  grade          CHAR(2),
  mark           DECIMAL(5,2) DEFAULT 0.0,
  PRIMARY KEY    (stdNo, courseID, semesterID),
  CONSTRAINT     fkRegisterStd FOREIGN KEY(stdNo) REFERENCES
        Student(stdNo) ON UPDATE CASCADE ON DELETE NO ACTION,
  FOREIGN KEY(courseID) REFERENCES Course(courseID)
        ON UPDATE CASCADE ON DELETE NO ACTION
)
```



ALTER TABLE

- ALTER TABLE modifies a table definition by altering, adding, or dropping columns and constraints

- Basic Syntax:

ALTER TABLE TableName
[ADD [COLUMN] columnName dataType [NOT NULL] [UNIQUE]
[DROP [COLUMN] columnName [RESTRICT|CASCADE]]
[ADD [CONSTRAINT [ConstraintName]] tableConstraintDefinition]
[DROP CONSTRAINT ConstraintName [RESTRICT|CASCADE]
[ALTER [COLUMN] {SET DEFAULT defaultOption | DROP DEFAULT}]

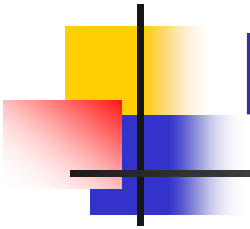
Examples: ALTER TABLE

- ALTER TABLE Staff
  ALTER position DROP DEFAULT;

- ALTER TABLE Staff
  ALTER sex SET DEFAULT ‘M’;

- ALTER TABLE PropertyForRent
  DROP CONSTRAINT StaffNotHandlingTooMuch;

- ALTER TABLE Client
  ADD prefNoRooms INT;

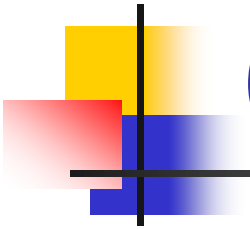


DROP TABLE

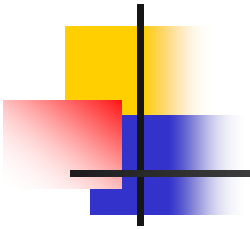
- Syntax:
  - DROP TABLE TableName
- DROP TABLE cannot be used to drop a table that is referenced by a FOREIGN KEY constraint.
- The referencing FOREIGN KEY constraint or the referencing table must first be dropped.
- Example

```
DROP TABLE PropertyForRent
```

Data Manipulation Language (DML)



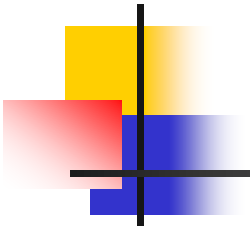
- SQL's Data Manipulation Language has facilities for modifying data as well as querying:
  - INSERT – adds new rows to a table
  - UPDATE – modifies existing data in a table
  - DELETE – removes rows of data from a table
  - SELECT – used for querying data



INSERT statement

- Two forms of INSERT statement
  - Inserting a single row
    - INSERT INTO TableName [(column-list)]
      VALUES (dataValueList)
  - Inserting multiple rows
    - INSERT INTO TableName [(column-list)]
      SELECT statement*

    Or:
    - INSERT INTO TableName [(column-list)]
      VALUES (dataValueList), (dataValueList),...



INSERT Examples

- Without specifying columns

  INSERT INTO Semester VALUES(1, 1, 2006)

  * Assumes an order of columns in CREATE TABLE clause

- Specifying column names

  INSERT INTO Student (stdNo, login, lastName) VALUES ('S001', 'STD928', 'Daniel');

  * Unspecified columns in the INSERT will have the DEFAULT value if specified, or the NULL value

UPDATE statement

- Syntax:

```
UPDATE  TableName
SET     columnName1 = dataValue1
        [, columnName2 = dataValue2...]
[WHERE  searchCondition]
```

  - *TableName* can be name of a base table or an updatable view.
  - SET clause specifies names of one or more columns that are to be updated
  - WHERE clause is optional:
    - If omitted, all rows are updated
    - if specified, only those rows that satisfy *searchCondition* are updated.
  - New *dataValue(s)* must be compatible with data type for corresponding column

UPDATE Examples

- Give all staff a 3% pay increase.

```
UPDATE Staff
SET salary = salary*1.03;
```

- Give all Managers a 5% pay increase.

```
UPDATE Staff
SET salary = salary*1.05
WHERE position = 'Manager';
```

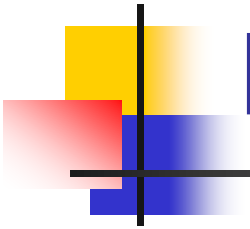
DELETE statement

- Syntax:

  DELETE FROM TableName

  [WHERE searchCondition]

  - *TableName* can be name of a base table or an updatable view.
  - WHERE clause is optional:
    - if omitted, all rows are deleted from table. This does not delete table.
    - If specified, only those rows that satisfy the *search_condition* are deleted



DELETE Examples

- Delete all viewings that relate to property PG4.

```
DELETE FROM Viewing
WHERE propertyNo = 'PG4';
```

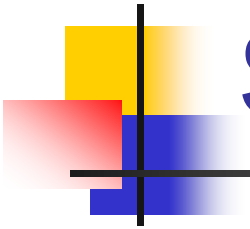
- Delete all records from the Viewing table.

```
DELETE FROM Viewing;
```

SELECT statement

- Syntax:

```
SELECT [DISTINCT | ALL]
    {* | [columnExpression [AS newName]] [,...] }
FROM        TableName [alias] [, ...]
[WHERE      condition]
[GROUP BY   columnList
[HAVING     condition]]
[ORDER BY   columnList]
```



SELECT statement (contd.)

order of executing these clauses

- FROM Specifies table(s) to be used.
- WHERE Filters rows.
- GROUP BY Forms groups of rows with same column value.
- HAVING Filters groups subject to some condition.
- SELECT Specifies which columns are to appear in output.
- ORDER BY Specifies the order of the output.

All Columns, All Rows

- List full details of all staff

```
SELECT staffNo, fName, lName, address,
       position, sex, DOB, salary, branchNo
FROM Staff;
```

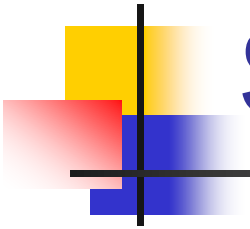
- Can use * as an abbreviation for 'all columns'

```
SELECT *
FROM Staff;
```

All Columns, All Rows (contd.)

Table 5.1 Result table for Example 5.1.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005



Specific Columns, All Rows

- Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary
FROM Staff;
```

Specific Columns, All Rows (contd.)

Table 5.2 Result table for Example 5.2.

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Use of DISTINCT

- List the property numbers of all properties that have been viewed.

```
SELECT propertyNo
FROM Viewing;
```

propertyNo
PA14
PG4
PG4
PA14
PG36

Use of DISTINCT (contd.)

- Use DISTINCT to eliminate duplicates:

```
SELECT DISTINCT propertyNo
FROM Viewing;
```

propertyNo
PA14
PG4
PG36

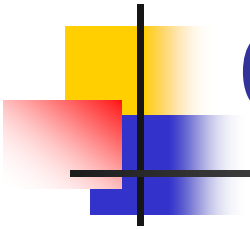
Calculated Fields

- Produce list of monthly salaries for all staff, showing staff number, first/last name, and salary.

```
SELECT staffNo, fName, lName, salary/12
FROM Staff;
```

Table 5.4 Result table for Example 5.4.

staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00



Calculated Fields (contd.)

- To name column, use AS clause:

```
SELECT staffNo, fName, lName,
                salary/12 AS monthlySalary
FROM Staff;
```

Comparison Search Condition

- List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000
```

Table 5.5 Result table for Example 5.5.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Compound Comparison Search Condition

- List addresses of all branch offices in London or Glasgow.

```
SELECT *
FROM Branch
WHERE city = 'London' OR city = 'Glasgow';
```

Table 5.6 Result table for Example 5.6.

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

Range Search Condition

- List all staff with a salary between 20,000 and 30,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

- BETWEEN test includes the endpoints of range.

Range Search Condition (contd.)

Table 5.7 Result table for Example 5.7.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

Range Search Condition (contd.)

- Also a negated version NOT BETWEEN.

- Could also write without BETWEEN as follows:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary>=20000 AND salary <= 30000;
```

- Useful, though, for a range of values.

Set Membership

- List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

Table 5.8 Result table for Example 5.8.

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Set Membership (contd.)

- Could have expressed set membership without IN, as follows:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position='Manager' OR
                position='Supervisor';
```

- IN is more efficient when set contains many values.
- There is a negated version (NOT IN).

Pattern Matching

- Find all owners with the string ‘Glasgow’ in their address.

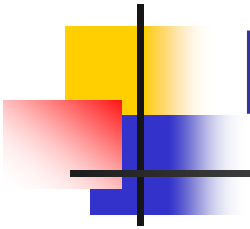
```
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE ‘%Glasgow%’;
```

Table 5.9 Result table for Example 5.9.

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

Pattern Matching (contd.)

- SQL has two special pattern matching symbols:
  - %: sequence of zero or more characters;
  - _ (underscore): any single character.
- LIKE '%Glasgow%' means a sequence of characters of any length containing '*Glasgow*'.
- LIKE '---' means a sequence of exact 3 characters.



NULL Search Condition

- List details of all viewings on property PG4 where a comment has not been supplied.
- Have to test for null explicitly using special keyword IS NULL:

```
SELECT clientNo, viewDate
FROM Viewing
WHERE propertyNo = 'PG4' AND
                comment IS NULL;
```

NULL Search Condition (contd.)

clientNo	viewDate
CR56	26-May-04

- Negated version (IS NOT NULL) can test for non-null values.

Single Column Ordering

- List salaries for all staff, arranged in descending order of salary.

```
SELECT staffNo, fName, lName, salary
    FROM Staff
    ORDER BY salary DESC;
```

Table 5.11 Result table for Example 5.11.

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

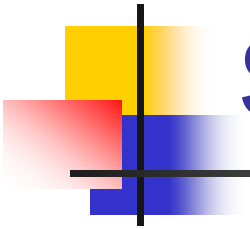
Multiple Column Ordering

- To arrange property for rent in ascending order of type and then by descending order of rent, specify attribute to order by from left-right in ORDER BY clause:

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type, rent DESC;
```

Table 5.12(b) Result table for Example 5.12 with two sort keys.

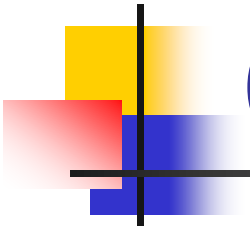
propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600



SELECT Statement - Aggregates

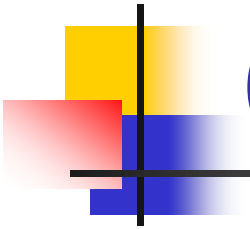
- ISO standard defines five aggregate functions:
  - COUNT returns number of values in specified column.
  - SUM returns sum of values in specified column.
  - AVG returns average of values in specified column
  - MIN returns smallest value in specified column.
  - MAX returns largest value in specified column.

SELECT Statement – Aggregates (contd.)



- Each operates on a single column of a table and returns a single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.

SELECT Statement – Aggregates (contd.)



- COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.
- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.

SELECT Statement – Aggregates (contd.)

- Aggregate functions can be used only in SELECT list and in HAVING clause.

- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot contain any column that is not an argument to an aggregate function. For example, the following is illegal:

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

Use of COUNT(*)

- How many properties cost more than £350 per month to rent?

```
SELECT COUNT(*) AS myCount
FROM PropertyForRent
WHERE rent > 350;
```

myCount
5

Use of COUNT(DISTINCT)

- How many different properties viewed in May 2004?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN '1-May-04'
        AND '31-May-04';
```

myCount
2

Use of COUNT and SUM

- Find number of Managers and sum of their salaries.

```
SELECT   COUNT(staffNo) AS myCount,
         SUM(salary) AS mySum
FROM Staff
WHERE position = 'Manager';
```

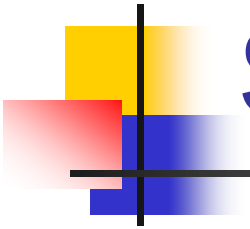
myCount	mySum
2	54000.00

Use of MIN, MAX, AVG

- Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS myMin,
       MAX(salary) AS myMax,
       AVG(salary) AS myAvg
 FROM Staff;
```

myMin	myMax	myAvg
9000.00	30000.00	17000.00



SELECT Statement - Grouping

- Use GROUP BY clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be *single-valued per group*, and SELECT clause may only contain:
  - column names
  - aggregate functions
  - constants
  - expression involving combinations of the above.

SELECT Statement – Grouping (contd.)

- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function. (But there may be columns in GROUP BY, but do not appear in SELECT list)
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

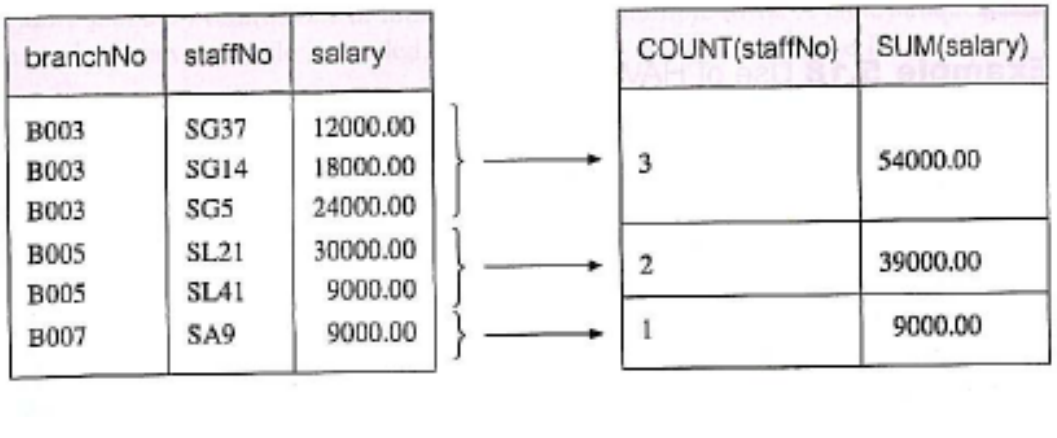
```
SELECT branchNo,COUNT(staffNo),SUM(salary)
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

Use of GROUP BY (contd.)

- Find number of staff in each branch and their total salaries.

```
SELECT      branchNo,
            COUNT(staffNo) AS myCount,
            SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

Use of GROUP BY (contd.)



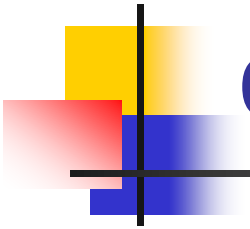
branchNo	staffNo	salary
B003	SG37	12000.00
B003	SG14	18000.00
B003	SG5	24000.00
B005	SL21	30000.00
B005	SL41	9000.00
B007	SA9	9000.00

COUNT(staffNo)	SUM(salary)
3	54000.00
2	39000.00
1	9000.00

```
SELECT          branchNo,
COUNT(staffNo) AS myCount,
                SUM(salary) AS
mySum
        FROM Staff
        GROUP BY branchNo
        ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

Restricted Groupings – HAVING clause



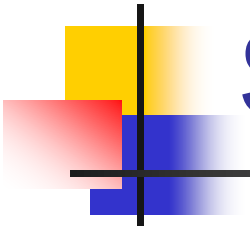
- HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas *HAVING filters groups*.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

Use of HAVING (contd.)

- For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

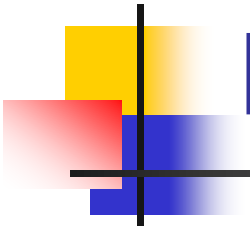
```
SELECT  branchNo,
        COUNT(staffNo) AS myCount,
        SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00



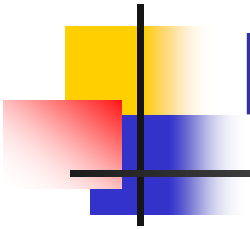
Summary

- SQL: Historical Perspective
- SQL: DDL
  - Integrity constraints and SQL
  - CREATE, ALTER and DROP
- SQL: DML
  - INSERT
  - UPDATE
  - DELETE
  - SELECT Basics



Lab This Week

- SQL query practice on two cases: Hotel and Registration



More on A2

- 1. Revise requirements and EER diagram in A1
- 2. Map the EER model to the relational model
- 3. Normalize the schema to BCNF
- Note: give 2 or 3 good examples of normalising tables