1B. SQL

Slides adapted from John Dowell

Structured Query Language (SQL)

- ISO standard way of using RDBMSs.
- Non-procedural (specifies what rather than how).
- Natural-language syntax resembling English sentences.
  - Keywords are case insensitive but CAPITALIZED as standard practice.
    - Data stored in database is case-sensitive.
  - Whitespace doesn’t matter.
- Despite the name, has specifications for both a DDL and a DML.
- Frequently used for querying *even in non-relational databases*, so useful to know.

Dialects

- SQL comes in different dialects that support different features.
- Focusing in this class on MySQL, which can be used with the phpMyAdmin graphical user interface.

Data definition

Database creation

- Create a new database:
  - CREATE DATABASE EstateAgent;
    - (Can optionally specify character set that data will be stored in)
- Make it the current database in use:
  - USE EstateAgent;

Table creation

- Create a table to hold data:
  - CREATE TABLE Staff (staffNo VARCHAR(4), fName VARCHAR(20), lName VARCHAR(20), position VARCHAR(20), sex VARCHAR(1), DOB VARCHAR (15), salary DECIMAL (5,2), branchNo VARCHAR(4));
- Specify table name and list out attributes – name and type; optionally, properties about the attributes.

Data types

- BOOLEAN (three values: TRUE / FALSE / UNKNOWN).
    - Last one comes up due to null values.
- CHAR(#) – fixed length string (CHAR = CHAR(1)).
- VARCHAR(#) – strings not necessarily the same length in storage, w/ upper limit on length.
- BIT [VARYING](#) – binary strings.
- INT; DECIMAL – exact numeric values.
- FLOAT – approximate numeric values.
- DATE (YEAR/MONTH/DAY), TIME (HOUR/MINUTE/SECOND), TIMESTAMP (DATE + TIME), INTERVAL.
- CHARACTER LARGE OBJECT, BINARY LARGE OBJECT.

Data types: default values

- After specifying data type, can optionally set a default value:
  - expirationDate DATE DEFAULT (CURRENT_DATE + INTERVAL 21 DAYS)

Integrity constraints

- Allows you to specify properties about the data you would like the database to guarantee is true at all times.
- Five types:
  - Required data,
  - Domain constraints,
  - Entity integrity,
  - Referential integrity,
  - General constraints.

Integrity constraints: required data

- A field cannot contain NULL values (missing data).
- Specify this property during table creation by adding ‘NOT NULL’ to an attribute.

Integrity constraints: entity integrity

- Can specify a primary key, which uniquely identifies a record in a table (imposes uniqueness and non-null requirements).
    - Use ‘PRIMARY KEY’ keyword after an attribute.
    - Add ‘PRIMARY KEY (col list…)’ at the tail end of the attribute list.
        - Can also name this constraint by adding ‘CONSTRAINT ConstraintName’ before.

- Can specify non-primary key to be unique using UNIQUE keyword.

Integrity constraints: referential integrity

- Foreign keys are special columns in one table that link its data to an entry in a second table by referencing that entry’s primary key (from its own table).
  - Add ‘FOREIGN KEY (colName) REFERENCES NameOfParentTable

Foreign key options

- What happens when the row my foreign key is referencing is deleted or its primary key is edited?
- Can optionally specify actions to take using ‘ON {DELETE | UPDATE} {CASCADE | SET NULL | SET DEFAULT | NO ACTION}’
  - NO ACTION: Update/delete fails because dependents exist (default setting).
  - CASCADE UPDATE: This entry’s foreign key is changed to match new value.
  - CASCADE DELETE: This entry is also deleted, possibly triggering further action if something is referencing this entry.
  - SET NULL: This entry’s foreign key becomes null.
  - SET DEFAULT: This entry’s foreign key is set to the column’s default value (can be specified at table creation).

MySQL engines: InnoDB & MyISAM

- MySQL has different storage engines.
- Can set the storage engine for a table upon creation using "ENGINE = [INNODB, MYISAM]" flag.
- InnoDB is the default.
  - Implements foreign key feature and ACID properties for transactions.
  - Better than MyISAM for updating.
  - Worse than MyISAM for querying/searching.

Alter table definition

- Tables can be altered (‘ALTER TABLE TableName’) after the fact to:
  - ADD COLUMN colName type
  - DROP COLUMN colName
  - ADD CONSTRAINT constraintName … (details)
  - DROP CONSTRAINT constraintName
  - ALTER colName {SET | DROP} DEFAULT

Delete table

- Delete a table using DROP TABLE TableName.

Data manipulation

Four main DML functions

- INSERT, SELECT, UPDATE, DELETE.
  - (Create, read, write, destroy.)

Populating the database with data

- Data can be inserted from a file stored locally to the database (such as from a text file called staff.txt):
- SQL load data statement:
  - E.g. LOAD DATA LOCAL INFILE “staff.txt" INTO TABLE staff;
- phpMyAdmin – use tools under the ‘Import’ tab.

Manual insertion of a row into a table

- INSERT INTO Staff(staffNo, fName, LName, position, sex, DOB, salary, branchNo)
  VALUES ('SG17', 'Joe', 'Brown', 'Manager', 'M', '12-09-90', 21000, 'B003’)

Update data in a table

- Give all staff a 3% pay increase.
    - UPDATE Staff
      SET salary = salary*1.03
- Give all Managers a 5% pay cut.
    - UPDATE Staff
      SET salary = salary * 0.95
      WHERE position = 'Manager'

Delete data from a table

- DELETE FROM Staff WHERE staffNo= 'SG17’
  - Need WHERE to limit the scope of this action.
- DELETE FROM Staff
  - This deletes all rows in a table(!) while not deleting the table itself.

Components of a query to retrieve data

- Retrieval queries are composed from a limited set of clauses:

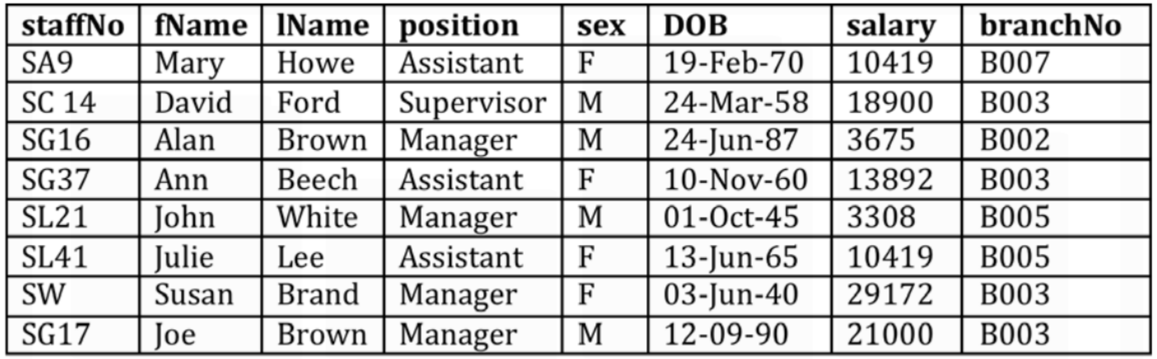
SELECT …	columns to appear in output
FROM …	table(s) to be used
WHERE …	filters rows by condition
GROUP BY …	forms groups of rows with same column value
HAVING …	filters groups subject to some condition
ORDER BY …	sorts the results

Example database

- Schema used in examples is for a fictional real estate company:
  - Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo);
  - Branch (branchNo, address, city);
  - Clients (clientNo, fName, lName, title, address, telNo, prefType);
  - Viewing (clientNo, propertyNo, viewdate, comment);
  - Property (propertyNo, address, type, rooms, rent, staffNo);

Simplest selection only uses SELECT and FROM

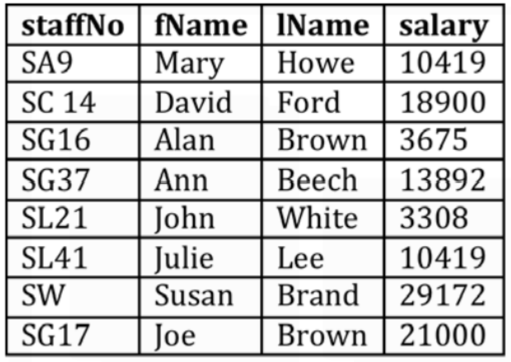
- *Task: List full details of all staff.*
  - SELECT staffNo, fName, lName, position, sex, DOB, salary, branchNo
    FROM Staff
  - SELECT * FROM Staff



staffNo	fName	lName	position	sex	DOB	salary	branchNo
SA9	Mary	Howe	Assistant	F	19-Feb-70	10419	B007
SC 14	David	Ford	Supervisor	M	24-Mar-58	18900	B003
SG16	Alan	Brown	Manager	M	24-Jun-87	3675	B002
SG37	Ann	Beech	Assistant	F	10-Nov-60	13892	B003
SL21	John	White	Manager	M	01-Oct-45	3308	B005
SL41	Julie	Lee	Assistant	F	13-Jun-65	10419	B005
SW	Susan	Brand	Manager	F	03-Jun-40	29172	B003
SG17	Joe	Brown	Manager	M	12-09-90	21000	B003

Selection: filter out columns

- *Task: 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



staffNo	fName	lName	salary
SA9	Mary	Howe	10419
SC 14	David	Ford	18900
SG16	Alan	Brown	3675
SG37	Ann	Beech	13892
SL21	John	White	3308
SL41	Julie	Lee	10419
SW	Susan	Brand	29172
SG17	Joe	Brown	21000

Use DISTINCT keyword to remove duplicates from results

- *Task: List the property numbers of all properties that have been viewed.*
  - SELECT DISTINCT propertyNo
    FROM Viewing;

SQL can apply mathematical operations to query results

- *Task: Produce a list of monthly salaries for all staff, showing staff number, first and last names, and salary details.*

    -
      ```
      SELECT staffNo, fName, lName, salary / 12
      FROM Staff;
      ```
- Calculated columns are new columns and can be renamed:

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

WHERE filtering: comparisons

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

  -
    ```
    SELECT staffNo, fName, lName, position, salary
    FROM  Staff
    WHERE salary > 10000;
    ```
- Other comparison operators: =, <>, <, >, <=, >=

WHERE filtering: compound conditions

- *Task: List addresses of all branch offices in London or Glasgow.*
  - SELECT *
    FROM Branch
    WHERE city = ‘London’ OR city = ‘Glasgow’;
- Other boolean operators: AND, OR, NOT

WHERE filtering: range

- *Task: 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;
    ```
- Ranges are inclusive: [20000, 30000]
- Can also use NOT BETWEEN
- Can implement using two >= or <= operators (multiple ways to skin a cat)

WHERE filtering: set membership

- *Task: List all managers and supervisors.*
  - SELECT staffNo, fName, lName, position
    FROM Staff
    WHERE position IN (‘Manager’, ‘Supervisor’);
- Can also use NOT IN
- Can implement using string of ORs

WHERE filtering: pattern-matching

- *Task: Find all properties with the string ‘awr’ in the street name.*

  -
    ```
    SELECT propertyNo
    FROM Property
    WHERE street LIKE '%awr%’
    ```
- Pattern is a regex with
  - %: wildcard; sequence of zero or more characters
  - _ (underscore): any single character.

WHERE filtering: NULL data

- *Task: List details of viewings where a comment has not been supplied.*
  - SELECT clientNo, viewDate
    FROM Viewing
    WHERE comment IS NULL
- Can also use IS NOT NULL

Sorting using ORDER BY

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

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

- DESC = descending
  ASC = ascending [default setting]

Sorting by multiple columns

- *Task: Produce an abbreviated list of properties in order of property type and rent.*
  - SELECT propertyNo, type, rooms, rent
    FROM Property
    ORDER BY type, rent DESC;

Aggregate functions

- Aggregation: calculation over a set of rows
- ISO standard defines five aggregate functions:
  - COUNT number of records (rows) in table incl. duplicates but not nulls
  - SUM total of values in specified column
  - AVG average of values in specified column
  - MIN smallest value in specified column
  - MAX largest values in specified column

Aggregate functions: details

- Operates on a set/group of values; returns single value
- COUNT, MIN, MAX for numeric and non-numeric fields
- SUM, AVG for numeric fields only
- Each aggregate function skips over nulls
- Include null values in count using COUNT(*)
- Remove duplicate values in count using DISTINCT keyword before column name

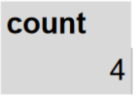
Using aggregate functions

- Aggregates can only be used in SELECT and HAVING clauses
- If an aggregate function is used in a SELECT clause and there is no GROUP BY clause, SELECT list cannot reference another column without using an aggregate function.
- Ex: illegal:
  - SELECT staffNo, SUM(salary)
    FROM Staff;

Using COUNT(DISTINCT)

- *Task: How many different properties were viewed in May 2012?*

  -
    ```
    SELECT COUNT( DISTINCT propertyNo ) AS count
    FROM Viewing
    WHERE viewDate
    BETWEEN '1-May-12’
    AND '31-May-12';
    ```

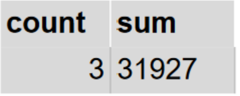


count
4

Using ‘COUNT’ and ‘SUM’

- *Task: Find the number of managers and sum of their salaries.*

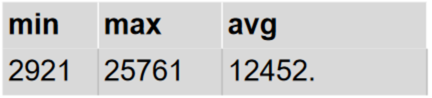
  -
    ```
    SELECT COUNT( staffNo ) AS count, SUM( salary ) AS sum
    FROM Staff
    WHERE position ='Manager'
    ```



count	sum
3	31927

Using ‘MIN’, ‘MAX’, ‘AVG’

- *Task: Find minimum, maximum, and average staff salary.*
  - SELECT MIN(salary) AS min, MAX(salary) AS max, AVG(salary) AS avg FROM Staff

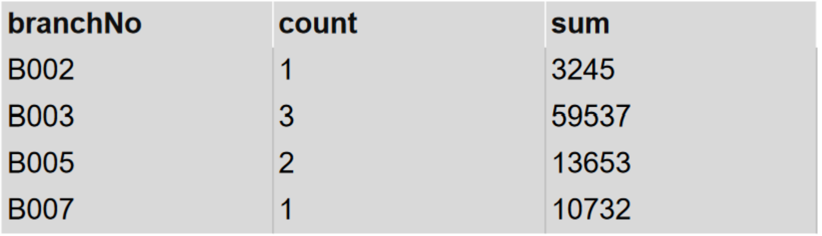


min	max	avg
2921	25761	12452.

GROUP BY: aggregation per category

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

  -
    ```
    SELECT branchNo, COUNT( staffNo ) AS count, SUM( salary ) AS sum
    FROM Staff
    GROUP BY branchNo
    ```



branchNo	count	sum
B002	1	3245
B003	3	59537
B005	2	13653
B007	1	10732

GROUP BY details

- Calculates subtotals per category.
- SELECT and GROUP BY closely integrated:
  - One item in SELECT result per group in GROUP BY.
  - Column names in SELECT list *not* enclosed in an aggregate function must appear in the GROUP BY clause.
- SELECT clause may only contain:
  - Column names,
  - Aggregate functions,
  - Constants,
  - An expression involving combinations of the above.

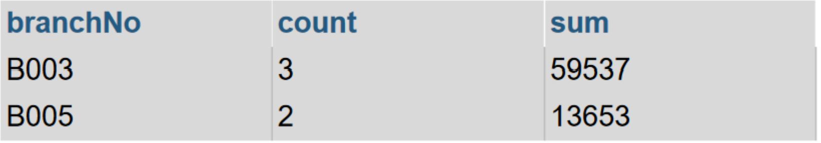
HAVING vs. WHERE

- WHERE filters out rows *before* aggregate functions [over groups] are calculated.
- HAVING filters out group entries in the final aggregated table.

Using HAVING

- *Task: 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 count, SUM( salary ) AS sum
    FROM Staff
    GROUP BY branchNo
    HAVING COUNT( staffNo ) > 1
    ORDER BY branchNo
    LIMIT 0 , 30
    ```



branchNo	count	sum
B003	3	59537
B005	2	13653

Subqueries

- SELECT queries can be nested within each other and within other DML functions (INSERT, UPDATE, DELETE).
- SELECT creates a temporary table: 0-D (scalar value), 1-D (single tuple/row or col), or 2-D (full table).
- Unlike aggregate functions, nested SELECTS can be inserted in WHERE / HAVING conditionals.

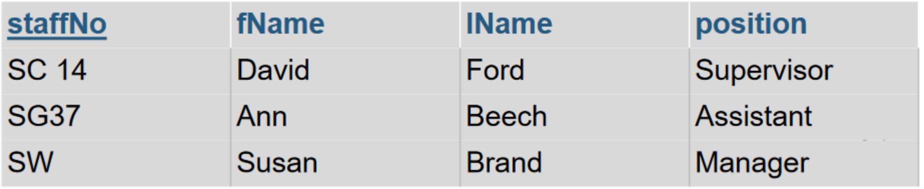
Subquery using equality conditional

- *Task: List staff who work in branch at ‘163 Main St’.*

  -
    ```
    SELECT staffNo, fName, lName, position
    FROM Staff
    WHERE branchNo = (SELECT branchNo
                        FROM Branch
                        WHERE street = '163 Main St')
    ```

Produces 0-D (one-cell) table with a single branchNo

Final result



staffNo	fName	lName	position
SC 14	David	Ford	Supervisor
SG37	Ann	Beech	Assistant
SW	Susan	Brand	Manager

Subquery with aggregation

- *Task: List all staff whose salary is greater than the average salary.*
  - SELECT staffNo, fName, lName, position, salary
    FROM Staff
    WHERE salary > (SELECT AVG(salary) FROM Staff)

Can’t use aggregate functions in a WHERE; use SELECT instead

Subquery using ‘IN’

- *Task: List properties handled by staff at ‘163 Main St’.*

  -
    ```
    SELECT propertyNo, street, city, postcode, type, rooms, rent
    FROM Property
    WHERE staffNo IN (
          SELECT staffNo
          FROM Staff
          WHERE branchNo = (
                      SELECT branchNo
                      FROM Branch
                      WHERE street = '163 Main St’))
    ```
- Recall: “WHERE filtering: set membership” – 1-D results of SELECT queries constitute a set.

‘ALL’ and ‘ANY’ (‘SOME’)

- Used in comparisons to only select data for which the comparison is true for all or some values.
- Comparison will be performed for each value in a 1-D table (list of values).
- Edge case: empty result set -> ALL returns true, but ANY/SOME returns false.

Subquery using ‘ANY’/’SOME’

- *Task: Find staff whose salary is larger than salary of at least one member of staff at branch B003.*

  -
    ```
    SELECT staffNo, fName, lName, position, salary
    FROM Staff
    WHERE salary > SOME (
                        SELECT salary
                        FROM Staff
                        WHERE branchNo = 'B003')
    ```

Subquery using ‘ALL’

- *Task: Find staff whose salary is larger than salary of every member of staff at branch B003.*

  -
    ```
    SELECT staffNo, fName, lName, position, salary
    FROM Staff
    WHERE salary > ALL (
                        SELECT salary
                        FROM Staff
                        WHERE branchNo = 'B003')
    ```

‘EXISTS’ and ‘NOT EXISTS’

- Turns the result of a SELECT into true/false:
  - True if and only if there exists at least one row in result table returned by subquery.
  - False if subquery returns an empty result table.
- As (NOT) EXISTS checks only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.
  - Common for subqueries following (NOT) EXISTS to be of form: (SELECT * ...)

Using EXISTS on a subquery

- *Task: Find all staff who work in a London branch.*

  -
    ```
    SELECT staffNo, fName, lName, position
    FROM Staff s
    WHERE EXISTS (
                SELECT *
                FROM Branch b
                WHERE s.branchNo = b.branchNo AND city = 'London')
    ```

Temporary table names

Table in larger query is available in subquery!

First condition matches staff to branches, second filters for London

Multiple tables in FROM

- Specifying multiple comma-separated tables in a FROM clause will join them, giving you access to all columns and all combinations of rows from each table.
- Probably too many rows – need to filter using WHERE.

Two-table example

- *Task: List names of all clients who have viewed a property along with any comment supplied.*

  -
    ```
    SELECT Client.clientNo, fName, lName, propertyNo, comment
    FROM Client, Viewing
    WHERE Client.clientNo = Viewing.clientNo
    ```

- Can use aliases for table names:

  -
    ```
    SELECT c.clientNo, fName, lName, propertyNo, comment
    FROM Client c, Viewing v
    WHERE c.clientNo = v.clientNo
    ```

Other ways of joining

- Using FROM… JOIN… ON
  - SELECT …
    FROM Clients AS c
    JOIN Viewings AS v ON c.clientNo = v.clientNo
  - Flexible; can use any condition after ON.
- FROM… JOIN… USING
  - SELECT …
    FROM Client JOIN Viewing USING clientNo
  - Succinct; assumes the tables both have columns with that name and you want them to be the same.

Three-table example

- *Task: For each branch, list staff who manage properties, including city in which branch is located and properties they manage.*

  -
    ```
    SELECT b.branchNo, b.city, s.staffNo, fName, lName, propertyNo, type
    FROM Branch b, Staff s, Property p
    WHERE b.branchNo = s.branchNo AND s.staffNo = p.staffNo
    ORDER BY b.branchNo, s.staffNo;
    ```

A note on joins

- By default, joins are *inner* joins, meaning they only contain entries from the tables where the condition was fulfilled.
- Can also specify *outer* joins, which retain rows in one or both tables even if they didn't match with anything.
  - Missing attributes that would have been filled by the match(es) in the other table are filled with nulls.
  - Left join / left outer join preserves all rows from the first table.
  - Right join / right outer join preserves all rows from the second table.
  - Full outer join does both.

Outer join example

- Task: List branches and properties in same city, along with unmatched [branches | properties | branches or properties].

  -
    ```
    SELECT b.*, p.*
    FROM Branch b [LEFT | RIGHT | FULL] JOIN Property p
    ON b.bCity = p.pCity;
    ```

Set operations on tables: union, intersect, and difference (‘EXCEPT’)

- Tables must be *union compatible* (have attribute lists that have matching data types (domains)).
  - Union (‘UNION’) creates a table that contains rows in one or both tables.
  - Intersection (‘INTERSECT’) creates a table the only contains rows in both tables.
  - Difference (‘EXCEPT’) creates a table like the first table, minus any rows also found in the second table.
- Options:
  - Use keyword ALL to allow duplicates.
  - Use keyword CORRESPONDING BY (col list…) to restrict table / scope of operation to a (list of) column(s) – not supported in MySQL.

Union example

- *Task: List all cities where there is either a branch office or a property.*

  -
    ```
    (SELECT city FROM Branch)
    UNION
    (SELECT city FROM Property);
    ```

  -
    ```
    (SELECT * FROM Branch)
    UNION CORRESPONDING BY city
    (SELECT * FROM Property);
    ```

Intersect example

- *Task: List all cities where there is both a branch office and at least one property.*
  - (SELECT city FROM Branch)
    INTERSECT
    (SELECT city FROM Property);
  - (SELECT * FROM Branch)
    INTERSECT CORRESPONDING BY city
    (SELECT * FROM Property);

Except example

- *Task: List all cities where there is a branch office but no properties.*

  -
    ```
    (SELECT city FROM Branch)
    EXCEPT
    (SELECT city FROM Property);
    ```

  -
    ```
    (SELECT * FROM Branch)
    EXCEPT CORRESPONDING BY city
    (SELECT * FROM Property);
    ```

Advanced data definition

Integrity constraints: domain constraints

- Specify using CHECK after an attribute when creating table:
  - CHECK (sex IN (‘M’, ‘F’))
  - CHECK (branchNo IN (SELECT branchNo FROM Branch))
- Create a new domain (custom data types that are defined separately from table construction and then usable):
  - CREATE DOMAIN MyDomain AS CHAR DEFAULT ‘M’ CHECK (VALUE IN (‘M’, ‘F’))
  - DROP DOMAIN [RESTRICT | CASCADE]

Integrity constraints: general constraints

- Specify using CHECK again.

- Create a new assertion (custom data types that are defined separately from table construction and then usable):

  - CREATE ASSERTION AssertionName CHECK (condition)

Creating views

- Views allow users to get a facet of the database helpful to their specialized needs (see: subschemas).
  - You probably won’t need them due to small database size.
- Create them from a query (SELECT):
  - CREATE VIEW MyView [(opt. renamed cols list)] AS SELECT …
    - Can also specify how to handle scenario where an update to a row causes it to fall out of the view (if it has a WHERE clause)
  - DROP VIEW MyView
    - Can also specify how to handle deletion of views derived from this one

Views in SQL

- Can be convenient, simplifying, have security uses, etc.
- Limitations on the types of queries and updates that can be performed on a view because it is based on a SELECT.
- Performance overhead.
- Implemented either as a query that runs every time the view is manipulated, or as a separate temporary table that must be reconciled with the original(s) every so often.

Creating an index

- Creates a data structure allowing faster lookups of data.
- Makes ordered presentation or random access of a certain attribute instantaneous.
  - CREATE [UNIQUE] INDEX IndexName ON TableName (col or col list) [ASC | DESC]

Transactions

- Can specify that a bundle of SQL statements should be committed or rolled back as a set (atomically).
- Use ‘COMMIT’.
- Can specify beforehand how much the set of statements interact with each other.

Authorization

- SQL allows permissions to be set for users on parts of the database:
  - GRANT {ALL PRIVILEGES or a comma-separated list of SELECT, INSERT, UPDATE, DELETE, USAGE, REFERENCES} ON NameOfDatabaseTableEtc TO {PUBLIC or list of accounts}
  - REVOKE {privileges} ON ObjectName FROM {account(s) | PUBLIC}

Summary

- We’ve covered:
  - Data definition: creating tables with attributes.
  - Data manipulation: inserting, updating, and deleting rows; selection selection selection.
  - Data definition: constraints on data, views, security, etc.