

2.1.03. QBE text-format example

- P. print, I. insert, D. delete, U. update
- _VARNAME, copy field value into variable

Employee	Title	FirstName	Surname	NINO	Salary
	Mr	Fred	Bloggs	P.	_starting
Employee	Title	FirstName	Surname	NINO	Salary
I.	Ms	Sahika	Jones	JM12240B	_starting

2.1.05. SQL

- Structured Query Language
 - (SQL or SEQUEL)
 - [Wikipedia reference](#)
- Success of relational databases
- Developed for SystemR at IBM
- ANSI standardised
- SQL-1986 (SQL1), ongoing extension
- SQL-1992 (SQL2), current version (Oracle 9i)
- SQL-1999 (SQL3), regular expression matching, recursive queries
- SQL-2003, XML features, auto-generated columns

2.1.05b. SQL command syntax

- Where follows here is a brief summary
 - Oracle syntax
 - Similar but not identical to MySQL/MSSQL
- General familiarity
- Query writing best learnt by doing it
- Lecture live-example
- Coursework 1 will be SQL
- [Oracle \(9i\) SQL reference](#)
- [MySQL \(5.0\) SQL reference](#)

2.1.05c. SQL in application

- Keyword oriented language
- Keywords not congruous with Relational model
- Lots of different ways to write SQL
 - Analogous to C/Java formatting
 - if (b==2) { a=1; } else { a=0; }
- Recommend using case to differentiate attributes and keywords
 - SELECT colour, size, shape FROM fruit WHERE weight>22;
- Oracle user accounts on Teaching database
- Namespace references, e.g. shared.cars

2.1.06. SQL Create schema

- Data definition commands
- CREATE
 - SCHEMA <schema_name> AUTHORIZATION <a>
 - or workspace
- Beware of names
 - Name collisions produce odd behaviours
- SQL Schema embraces Tables (relations), constraints, views, domains, authorizations

2.1.07. SQL Create table

- CREATE TABLE
 - <schema_name>.<relation_name>
 - (
 - <attribute_definitions>
 - <key>
 - <constraints>
 -)
- CREATE TABLE example (Oracle)
- Tables can (and should) be indexed by user
- e.g. <username>.<tablename>
- Normal login implies username
- Non-local table access

2.1.08. Data types and domains (Oracle)

- Numeric
 - ENUM
 - NUMBER, NUMBER(i), NUMBER(i,j)
 - Formatted numbers, i precision, j scale
 - (number of digits total, after decimal point)
 - Character-string

- CHAR(n) - n is length
 - VARCHAR2(n) - n is max
- DESCRIBE output example
- Multi-database comparison of Datatypes
- Database legacy: limited storage necessitated efficient storage
- Does it need to be efficient anymore?

You might consider all SQL types as being conceptually similar to attribute types in the relational model, although in reality the implementation of these types in a DBMS only approximates the mathematical purity of unordered domain sets etc.

2.1.08b. Data types and domains (MySQL)

- Numeric
 - TINYINT, INT, INT UNSIGNED
 - FLOAT, DOUBLE, DECIMAL
 - ENUM
 - Character-string
 - CHAR(n) - n is length
 - VARCHAR(n) - n is max
 - TINYTEXT, TEXT
 - Beware different default/maximum lengths to Oracle
 - BLOB
 - Multi-database comparison of Datatypes

2.1.09. Time-based data types

- Date and Time
 - DATE
 - Ten positions, components YYYY-MM-DD
 - TIME
 - Eight positions, components HH:MM:SS
 - TIME(i)
 - Time fractional seconds precision
 - Adds i+1 positions
 - TIMESTAMP
 - optionally WITH TIME ZONE
 - Very sensitive to syntactical ambiguities
 - day/month/year/hour/minute separators

2.1.10. DROPIng

- DROP <object> <obj_name> <flags>
- DROP SCHEMA <schema_name> CASCADE
 - drops all workspace tables, domains
- DROP TABLE <relation_name> RESTRICT

- only drops table if
- not referenced in any constraints/views
- Notion of cascading
- Table links

2.1.11. ALTERing

- Schema evolution
- Design side
- ALTER TABLE <schema_name>.<relation_name> ADD <var_name> <var_type>;
- Example
 - ALTER TABLE uni.student ADD hall VARCHAR(32);
- Upper and lower case syntax
- Naming conventions

2.1.12. Queries

- Helper interfaces
 - HeidiSQL/phpMyAdmin/Sword/SQLplus
 - Design/perform a lot of routine queries for you
 - Important to learn SQL, reinforcement
 - Designing select queries is more difficult
 - Visual interfaces still lacking in this area
- Select queries in SQL
- Basic singlets
- Renaming
- Queries with Joins
- Nested queries

2.1.13. SQL Queries

- SELECT statement
- Similar to relational data model SELECT then PROJECT
 - SELECT <attribute list>
 - FROM <table list>
 - WHERE <condition>;

ID	Make	Model	Derivative	h
1	AC	Ace	3.5 Twin Turbo roadster	3
2	AC	Aceca	3.5 Twin Turbo coupe	3
3	AC	Cobra	5.0 Mk IV CRS roadster	2
4	AC	Superblower	5.0 V8 roadster	3
5	AC	Superblower	5.0 V8 Spirit of Brookla	3
6	Alfa Romeo	147	1.6 16V TS Turismo 3-c	1
7	Alfa Romeo	147	1.6 16V TS Lusso 3-do	1
8	Alfa Romeo	147	2.0 16V Selespeed Lus	1
9	Alfa Romeo	147	2.0 16V Lusso 3-door	1
10	Alfa Romeo	147	1.6 16V TS Turismo 5-c	1
11	Alfa Romeo	147	1.6 16V TS Lusso 5-do	1
12	Alfa Romeo	147	2.0 16V Lusso 5-door	1
13	Alfa Romeo	147	2.0 16V Selespeed Lus	1
14	Alfa Romeo	156	1.6 TS Lusso	1
15	Alfa Romeo	156	1.6 TS Veloce (Leather	1
16	Alfa Romeo	156	1.8 TS Lusso Saloon	1
17	Alfa Romeo	156	1.8 TS Veloce (Recaro)	1
18	Alfa Romeo	156	1.8 TS Veloce (Leather	1
19	Alfa Romeo	156	2.0 TS Lusso Saloon	1
20	Alfa Romeo	156	2.0 TS Veloce (Recaro)	1
21	Alfa Romeo	156	2.0 TS Veloce (Leather	1
22	Alfa Romeo	156	2.0 TS Veloce (Leather	1

2.1.14. SQL Queries

- SELECT <attr_list>
 - FROM R,S,T
 - WHERE DNO = 10
- equivalent to
- P<attr_list>(SDNO=10 (R X S X T))
- True-false evaluation tuple by tuple
- WHERE clause as compound logical statement

2.1.15. SQL Queries

- Produces a relation/set of tuples
- Can be used to extract a single tuple
- e.g. SELECT bday, age
 - FROM student
 - WHERE fname='Tim' AND lname='Smith'
 - Result = (13-05-80, 20)
- Argument quoting (')
 - SQL poisoning
 - Not null
 - Not numeric values
- MySQL Attribute quoting (`)
 - Hypothetical attribute `all`, all, and ALL

SQL poisoning is a vulnerability exposed by inadequate escaping of arguments/variables used to compose

SQL queries.

E.g. *Tim* in previous example, could be *Tim'*; *DELETE FROM student;* *SELECT * FROM student WHERE 1*

2.1.16. Renaming and referencing

- AS keyword
- (Partial) Attribute renaming in projection list
 - SELECT fname AS firstName, minit, lname AS surname...
- Role names for relations
 - SELECT S.FNAME, F.FNAME, S.LNAME
 - FROM STUDENT AS S, STUDENT AS F
 - WHERE S.LNAME=F.LNAME
- (Total) Attribute renaming in FROM
 - SELECT s.firstName, s.surname
 - FROM student AS s(firstName,surname,DOB,NINO,tutor)
- Wildcards (SELECT s.* FROM...)

2.1.17. SQL Tables

- Relations are bags, not sets
 - e.g. projection of non-key attributes
- Set cannot contain duplicate item/repetition
- Duplicates exist in bags and be:
 - SELECT DISTINCT (eliminated)
 - SELECT ALL (ignored/kept)

2.1.18. Queries and Joins

- Relational database allows inter-related data
- SQL select FROM gives Cartesian product
- WHERE clause defines join condition
 - SELECT proj.pnum, mgr.ssn
 - FROM project AS proj, employee AS mgr
 - WHERE proj.mgrssn = mgr.ssn;
- Alternatively, explicitly define join (note type)
 - SELECT project.pnum, employee.ssn
 - FROM project INNER JOIN employee
 - ON project.mgrssn = employee.ssn;

2.1.18b. Outer joins

- Outer joins are crucial in the real-world

- Databases often contain NULLs (3VL)
- Analysis of where the crucial data is across a relationship
- Previous example, only get project data for managed projects
 - SELECT project.*, employee.*
 - FROM project INNER JOIN employee
 - ON project.mgrssn = employee.ssn;

Project	ID	Name	mgrssn		
	1	Marketing	1		
	2	Development	NULL		
	3	Website	4		
Employee	SSN	Firstname	Surname		
	1	Alex	Jones		
	2	Jamal	Al Hak		
	3	Nissan	Imdana		
	4	Johan	Fredriksson		
InnerJoin	ID	Name	SSN	First	Surname
	1	Marketing	1	Alex	Jones
	3	Website	4	Johan	Fredriksson

2.1.18c. Outer joins (cont)

Scale of loss isn't always instantly obvious

NULLs often used unpredictably

May want project information, even if no employee attached as manager

- SELECT project.*, employee.*
- FROM project LEFT OUTER JOIN employee
- ON project.mgrssn = employee.ssn;

Project	ID	Name	mgrssn		
	1	Marketing	1		
	2	Development	NULL		
	3	Website	4		
Employee	SSN	Firstname	Surname		
	1	Alex	Jones		
	2	Jamal	Al Hak		
	3	Nissan	Imdana		
	4	Johan	Fredriksson		
LeftOuter	ID	Name	SSN	First	Surname
	1	Marketing	1	Alex	Jones
	2	Development	NULL	NULL	NULL
	3	Website	4	Johan	Fredriksson
FullOuter	ID	Name	SSN	First	Surname
	1	Marketing	1	Alex	Jones
	2	Development	NULL	NULL	NULL
	3	Website	4	Johan	Fredriksson
	NULL	NULL	2	Jamal	Al Hak
	NULL	NULL	3	Nissan	Imdana

2.1.19. 2y and 3y joins

- Queries can encapsulate any number of relations
 - Even one relation many times (in different roles)
- Relationship chain
- Across many relations
 - Tuples as Entities OR Relationships
- e.g. Employee -> Works_on -> Project -> Department -> Manager

2.1.20. Recursive closure

- Can't be done in SQL2
- Recursive relationships
- Unknown number of steps
- SQL2 can't generalise in single query

2.1.21. Nested queries

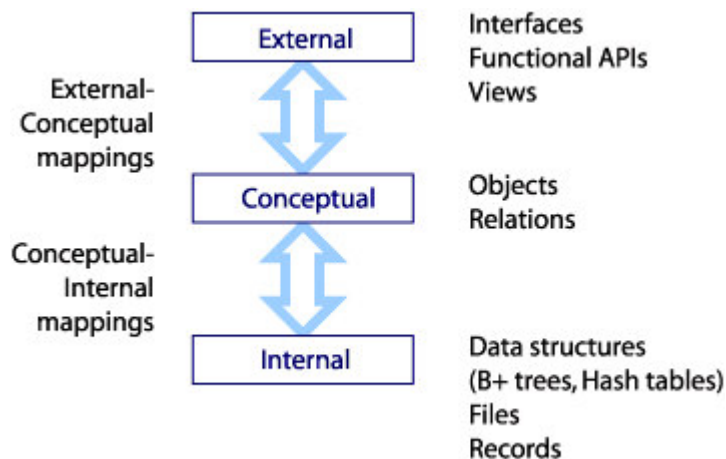
- Essential one or more (inner) queries within an (outer) query
- Inner and outer query
- Not to be confused with inner and outer joins
- Inner query can go in three places
 - SELECT clause (projection list)
 - Must return a single value, then aliased as attribute in outer result
 - FROM clause
 - Inner query result used as standard table in FROM cross product
 - WHERE clause

2.1.21b. Nexted query example

- Use of query result as comparator for other (outer) query
 - SELECT DISTINCT course
 - FROM dept WHERE course IN (
 - SELECT d.course
 - FROM dept AS d, faculty AS f, student AS s
 - WHERE d.ownfac=f.id AND s.owndep=d.id
 - AND f.name='Eng' AND s.year='3'
 -) OR course IN (
 - SELECT course
 - FROM dept
 - WHERE code LIKE 'COMS3%');

2.1.22. Bridging SQL across 3 tiers

- Three tier database design
- Changing role of DBMS
- Indices
- Aggregate functions (conceptual)
 - Over bags and sub-bags
- Creating and updating views (ext)
- SQL embedding



In this subsection we look at the different roles SQL play across the three tiers of database design. We discuss the areas in which SQL is lacking and how those deficiencies can be complemented by embedding SQL in other languages.

2.1.25. Indices

- Low/Internal level
- Index by one attribute
- For queries selecting by that attribute:
 - Faster tuple access (ordered tuples)
 - Reduces database memory load
 - Small cross product relation, only crosses requisites
 - Accelerates query resolution time
- `CREATE INDEX Index_Name ON RELATION(Attribute);`

2.1.26. Aggregate functions

- Run over groups of tuples
- Takes a projected attribute list as an argument
- Produce relation with single tuple
- SUM, MAX, MIN, AVG, COUNT
- e.g. AggFunc over all tuples
 - `SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY)`

- FROM EMPLOYEE;
- Single attribute lists (distinct values)
- Multi-attribute lists (granularity of distinct values by pairing)

2.1.27. Aggregates over sub-bags

- Can run over subsets of tuples
- GROUP BY keyword
- Specifies the grouping attributes
- Need to also appear in projected attr_list
- Show result along side value for group attr
- e.g. AggFunc over subgroups
 - SELECT dno, COUNT(*)
 - FROM employee
 - GROUP BY dno

Quick SQL check, do all attributes in the SELECT projection list appear in the GROUP BY projection list.

2.1.28. Creating views

- Views are partial projections
- Virtual relations, or views of live relations
- Update synchronised
 - CREATE VIEW <virtual_relname>
 - AS <real_relation>
- Real relation could be a query result
- Clever bit is the change propagation
- UPDATES made to the view dataset are flooded back to relations
 - INSERT and DELETE behaviour needs to be defined
 - Non-trivial as INSERT into view (virtual relation) may leave holes in real relation

2.1.29. Embedding SQL

- SQL (alone) can do lots of clever things in one expression
- But can only execute a single expression
- Can structure SQL commands into proper programming languages
- Java Database Connection (JDBC)
 - javac, then java VM
- COBOL, C or PASCAL
 - precompiled with PRO*COBOL or PRO*C
- Procedure Language (PL/SQL)
 - Oracle/MySQL procedural language
 - *Stored* procedures can take parameters